12-19-2018 11:00 AM

Partitioning and Offloading for IoT and Video Streaming Applications that Utilize Computing Resources at the Network Edge

Navid Bayat
The University of Western Ontario

Supervisor
Lutfiyya, Hanan
The University of Western Ontario

Graduate Program in Computer Science
A thesis submitted in partial fulfillment of the requirements for the degree in Doctor of Philosophy

Abstract

The Internet of Things (IoT) is a concept in which physical objects embedded with sensors, actuators, and network connectivity can communicate and react to their surroundings. IoT applications connect physical objects for the purpose of decision making by sensing and analysing generated data from the embeddedsensors in physical objects. IoT applications are growing rapidly as sensors become less expensive. Sensors generate large amounts of data that may meaningless unless the data is used to derive knowledge with in a certain period of time. Stream processing paradigm is used by IoT to provide requirements of IoT applications. In a stream processing paradigm, unlike traditional data bases, data is not stored but rather processed as it is generated. To transfer generated data from distributed data sources to a processing center such as cloud may not allow for real-time processing due to the network delay. Another high-demand application is live streaming of video. The performance of live video stream systems is inferior when there is a sudden large demand in the number of users. This thesis addresses some of the limitations of current architectures for video streaming systems and IoT applications based on the use of nearby computing resources (e.g., cloudlet, fog).

First, we addressed the degrading performance in video stream systems when a flash crowd occurs. The performance of video streaming systems is affected by flash crowd and degrade the quality of service for subscribers to the content delivery system. A flash crowd happens when there is a sudden large increase in the number of users. Therefore, flash crowds increase network traffic for any particular server. The main challenge is to make sure that the video streaming system has sufficient capacity to handle the occurrence of flash crowds.

Second, we address the limitation of current architectures for running mobile applications by introducing a dynamic partitioning and offloading of a mobile application. Mobile devices have limited resources including short battery life, storage capacity and processor performance. This limits the applications that can run on it. Mobile applications can be partitioned so that some of the application runs on a cloud. This works well for applications with relatively little data to be transferred and that do not have a high level of interaction with the user. Challenges with applications that have large amounts of data to be transferred and have a high level interactiveness is the high latency incurred by the network and packet loss of the wireless network. A mobile application can be partitioned so that part of it runs on a nearby computing resource e.g., fog node or cloudlet. This thesis presents a framework that introduces fine-grained offloading approach and support for runtime and dynamic partitioning of an application.

Third, we present a solution for placement of stream operators over distributed fog nodes for live processing of data streams from geographically distributed data sources. This placement of stream operators takes place in such a way that it supports applications with a high volume of data that require real-time (or near real-time) analysis To this end, this thesis proposed a set of algorithms for placement of stream operators among fog nodes.

Keywords: Internet of Thins, Data Streaming, Big Data, Context-aware, Query Graph, Fog Platform, Offloading, Partitioning.

Acknowlegements

Firstly, I would like to express my sincere gratitude to my advisor Professor Hanan Lutfiyya for her continuous support of my Ph.D study and my research, for her patience, and motivation. Her guidance helped me in all the time of my research and writing of my thesis. I could not have imagined having a better advisor for my Ph.D program. Last but not least, I would like to thank my parents and to my brother for supporting me spiritually throughout my Ph.D study and writing my thesis.

Contents

List of Figures

List of Tables

List of Appendices

Chapter 1

Introduction

Over the last decade, the cloud paradigm has been extensively adopted [159]. A recent survey [178] on the adoption rates of cloud computing by enterprises reported that 63% of enterprises are using the cloud. The reason for the success of the cloud paradigm is that it offers scalability, flexibility, and services on demand to customers [137][8]. For example, the cloud paradigm provides services that include infrastructure as a service, platform as a service, and software as a service [148]. There are a number of commercial cloud platforms that provide computing resources to their clients over the Internet such as Amazon Web Services and Microsoft Azure [54]. A key advantage to cloud computing is that computing resources for an application can be adjusted on demand to support fluctuating workloads and thus drive down costs for enterprises [8]. Many smartphone applications use the cloud to offload resource intensive computational tasks. For example, Apple's Siri [151] performs computational intensive speech recognition in the cloud and returns the results to the user [81].

The rest of this chapter is organized as follows: Section 1.1 describes emerging trends in applications. Section 1.2 discusses the characteristics and requirements of Internet of Things (IoT) applications and real-time data-intensive applications. Limitations of current architectures for these applications is discussed in Section 1.3. The contribution of this thesis is explained in Section 1.4. Section 1.5 provides the structure of the thesis.

1.1 Internet of Things and Emerging Applications

The Internet of Things (IoT) is a paradigm that envisions the physical world as a smart space in which physical objects are equipped with sensors, actuators, and network connectivity that can communicate and react to their surroundings [13]. Cisco estimates that there will be around 25 billion to 50 billion connected devices by 2020 [64]. A new generation of applications is emerging in which data is collected from the sensors embedded into physical objects. Several of these applications are described below [108]:

- Early Disaster Alerting: Sensors can collect crucial information about the environment and detect environmental disasters such as earthquakes and tsunami which help save lives.

- Health Care and Patients Surveillance: Constant monitoring of patients can save lives.

For example, emergency care can be dispatched immediately when a patient begins to experience a heart attack.

- Smart Surveillance Camera: The use of smart surveillance cameras allow authorities to detect when crime has occurred and respond faster.

- Agricultural Efficiency: Soil moisture sensors and weather sensors can sense soil moisture and take weather information into account for smart irrigation systems. Smart irrigation systems only water crops when needed and thus reduce the amount of water usage.

- Smart Navigation Systems in Smart Cities: Consider a smart city where all the vehicles are equipped with a sensor that periodically emits a position report that identifies the vehicle's location. The position reports can be used to determine congestion. If there is congestion, vehicles can be notified so that drivers can reroute.

- Smart Buildings: Consider a scenario where a sensor network is used to monitor the temperature of rooms in a building to sound an alarm in the case of a fire. This allows for for immediate and necessary fire detection [141]. Each temperature measurement report generated by any sensor in the building should be checked as soon as possible in order to detect a fire.

In the near future, the number of sensors are expected to grow exponentially and hence the increase in IoT applications (with real-time or near real-time requirements) [171].

In the past decade, not only have high data rate sensors such as video cameras become more pervasive, but the consumption of video contents have also changed from offline viewing to online streaming [153]. Furthermore, as portable devices with cameras have become more ubiquitous, users are able to create and share videos. Users are increasingly demanding content at any time of day, from any device and from any place. With these media streaming applications, it is important that the user receives the media content in real-time. This is especially important for live events (e.g., sports events). For example, live streaming events such as video broadcasts from the Olympic Games can cause very high traffic due to a huge number of fans [185]. In addition, many security applications use video surveillance cameras to provide security. These security applications process the received video content from surveillance cameras to extract knowledge and deliver the results (suspicious activity) to the user(s) devices (e.g., smartphone) through the Internet [31].

1.2 Characteristics of Emerging Applications

IoT applications and media streaming applications (e.g., audio and video applications) are expected to put the Internet under tremendous pressure due to exponential growth in data [7]. These applications have one or more of these characteristics: (i) high processing requirements; (ii) low processing latency; (iii) the generated data should be processed within a small window of time before the data loses its value; (iv) high consumption of network bandwidth; and (v) large number of subscribers.

1.3 Limitations of Current Architectures

Currently, IoT applications assume that sensor data is sent to a cloud for analysis and to determine a response. For example, a Fitbit offloads its data to an online account maintained by services hosted on a cloud. This works well for a Fitbit since real-time analysis and response is not essential. However, for security applications that use video surveillance this is not the case. The problem of relying solely on the cloud is the issue of latency. Latency is due to the network delay from the IoT devices to the cloud, which may often times lead to poor quality of service (QoS). Furthermore, the network core can become overwhelmed [111]. This makes the use of a cloud unsuitable for applications that require a high level of interaction with users and applications with real-time decision making requirements [77][115][44][78][162].

Video streaming applications make use of peer-to-peer networks where processing often makes use of computing resources close to data sources. This is done to address latency. However, much of the existing work suffers from overutilization or underutilization of geographically distributed processing units due to lack of management information [78][162].

1.4 Thesis Focus

This thesis addresses some of the limitations of current architectures for video streaming and IoT applications by using nearby computing resources (e.g., cloudlet, fog). A cloudlet is defined as a resource-rich computer that is connected to the Internet and is available at the network edge for use by nearby end-devices (e.g., smartphones, tablets, vehicles) [82][165][154][98][23]. Fog (similar to a cloudlet) extends the cloud power to the edge of networks, in particular wireless networks for the Internet of Things (IoT) [89][30]. Fog nodes are located away from the main cloud data centers (e.g., at any point from the data source to the cloud) [30]. Differences between cloudlet and fog are discussed in [58]. The rest of this section presents the thesis contributions.

1.4.1 Video Streaming Applications

Peer-to-Peer (P2P) networks consist of a set of nodes used to share access to data. P2P networks provide a robust, reliable and scalable message routing and delivery on top of a physical (underlay) network. Large-scale commercial P2P streaming networks such as Coolstreaming [200] and PPLive [166] offer numerous video channels to thousands of users, simultaneously. These are referred to as multi-channel P2P streaming networks [186]. In these types of networks, peers can participate in more than one channel and switch between them. This thesis addresses the following challenges:

- Imbalance of resource usage of different channels
- Dealing with flash crowds which occur when a large number of peers attempt to join a popular channel

1.4.2 IoT Applications and Data Stream Processing

In IoT applications, data takes the form of data streams. A data stream is defined as an unbounded sequence of tuples (stream elements) that is produced incrementally over time [76] [134]. A tuple is defined as an atomic data item embedded in a data stream and can be processed by a processing unit [55]. IoT applications can use queries to aggregate continuous data from data sources and process streams of data to produce the output [198][9]. In an IoT application, a query may run continuously. This type of query is referred to as continuous query [14]. A continuous query consists of one or more query operator(s) such as map, join, aggregate, filter and union. A query operator is a function which is used to process an input data stream and produce output results [83][157].

This thesis proposes an approach for placing query operators on fog nodes and the cloud. In this thesis, a query operator placement algorithm is proposed, which supports:

- Real-time aggregating and analysing data streams from different geographically distributed data sources.
- Combining more operators into one query graph instead of having different query graphs for each operator helps to save bandwidth and reduce transmission costs.
- Optimized operator placement and operator reuse.
- Scalability for distributed data stream management systems.

1.5 Thesis Structure

The remainder of this thesis is organized as follows:

- Chapter 2: Chapter 2 discusses data stream processing for IoT applications background and peer-to-peer live video streaming systems.
- Chapter 3: Chapter 3 presents a novel framework for multi-channel P2P live video streaming that provides de-centralized mechanisms for handling flash crowds that includes incentive mechanism, load balancing mechanisms, and cross-channel help among peers for live video streaming in multi-channel P2P systems.
- Chapter 4: Chapter 4 presents a dynamic partitioning strategy that considers network conditions and the amount of data being transferred, shows how a cloudlet mesh can be used, and presents performance results that illustrate the advantages of a dynamic partitioning strategy and a cloudlet mesh.
- Chapter 5: Chapter 5 presents related work for concepts of data streaming such as operators and continuous queries.
- Chapter 6: In chapter 6 we formulate the query operator placement problem.

- Chapter 7: Chapter 7 presents the results of experimental studies that compare the use of using one or more levels of fog nodes for high and low volumes of data. Also, this chapter explores the need for an approach that places query operators on fog nodes.
- Chapter 8: Chapter 8 presents the proposed algorithms for query operator placement among distributed fog nodes.
- Chapter 9: Chapter 9 presents simulation results of the algorithms proposed in Chapter 8. The main goal is to show how the proposed algorithms can improve response time.
- Chapter 10: Chapter 10 summarizes the work in this thesis.

Chapter 2

Background on Peer-to-Peer Networks

This chapter presents background knowledge on work related to peer-to-peer (P2P) networks and streaming of live video over P2P networks.

This chapter is organized as follows: Section 2.1 presents peer-to-peer concepts and semantics. Section 2.2 provides an overview of basic peer-to-peer data streaming systems. Section 2.3 presents peer-to-peer overlay network typologies. Section 2.4 discusses the limitations of current peer-to-peer live video streaming systems.

2.1 Peer-to-Peer Networks

Peer-to-Peer (P2P) networks consist of a set of nodes (peers) that share their available resources (e.g., upload and download bandwidth, computational resources, data storage resources). P2P networks provide a robust, reliable and scalable message routing and delivery on top of a physical (underlay) network. Peer-to-Peer network applications such as video on demand, live video streaming, and file sharing have become popular due to their scalability [50]. Recent studies [185][195][42] have shown that the overall P2P traffic on the Internet has been constantly increasing and that more than 60% of Internet traffic is generated by P2P network applications .

2.1.1 Classification of P2P Networks

P2P networks can be classified into the following categories:

- Centralized P2P Networks: In a centralized P2P network, a set of servers maintain a database of available services. A newly joined peer in a centralized P2P network needs to submit a request for the resource (e.g., a file) to a server. The server then looks to find the list of peers that can provide the requested service e.g., the requested file.

- Decentralized P2P Networks: Unlike the centralized approach, in a decentralised P2P network, peers connect to other peers (neighbours) and share resources among each other without any centralized server. Each peer obtains a service by sending a request to all its neighbours. Two common types of decentralized P2P networks are unstructured and structured:

- Unstructured: In an unstructured P2P network, peers randomly establish a connection to a set of peers. The main problem with unstructured P2P networks is that a peer has to search the entire network to find a resource. The result is more latency and overhead to find a particular piece of data.
- Structured: In a structured P2P network, the overlay is organized into a specific topology (e.g., tree-based P2P network and mesh-based P2P network) [109].

2.2 P2P Streaming Network

P2P networks have become a popular approach to provide live video and video-on-demand (VoD) services. Commercial P2P live streaming and video-on-demand networks, such as PPLive [166], PPStream [176], UUSee [19], have successfully supported large subscribers [200].

2.2.1 P2P Live Streaming Systems

A P2P live stream includes a continuous flow of video that is generated in real-time. In a P2P live streaming network, a live video is disseminated to all subscribers in real-time. The video playbacks on all users are synchronized (i.e., each peer watches almost the same position of the video). In a P2P live streaming system, peers can join an on-going live streaming session and start watching the stream from the time they joined.

Recently, a new breed of commercial P2P video streaming applications has emerged [200] [166][25]. Large-scale commercial Peer-to-Peer (P2P) networks such as Coolstreaming [200] and PPLive [166] are extensively used for live video streaming. In almost all of these P2P video streaming systems, multiple channels broadcast video to thousands of users, simultaneously [48].

2.3 P2P Network Topologies

This section discusses P2P network topologies.

2.3.1 Tree-Based P2P Network for Live Video Streaming

In a tree-based system, peers are connected and organized into a tree where the server acts as the root. Each peer downloads from its parent and uploads to its children. Accordingly, tree-based systems use a push-based data-driven method [28][116]. The major drawbacks of the single-tree based approach is that the leaf peers cannot contribute their upload bandwidth to the tree. Therefore, to overcome with the aforementioned disadvantage, the multi-tree approach is introduced. In the multi-tree approach, a peer can join different streaming trees (in the multi-tree a peer can be a leaf in one tree but be a parent (internal node) in another tree).

2.3.2 Mesh-based P2P Network for Live Streaming Network

In mesh-based P2P live streaming networks, each peer exchanges resources (e.g., data) with its neighbours. The main advantage of mesh-based P2P live streaming networks over tree-based P2P live video streaming is that mesh-based P2P live streaming is robust against peer churn. In a mesh-based P2P live streaming, if a neighbour peer(s) of a peer leave the system, the peer can receive stream (or download the video) from the remaining neighbours.

2.4 Gap Analysis

The limitation of the current architecture is that it uses a centralized or semi-centralized approach to respond to flash crowds[149][114]. The limitation on the centralized approach is that it uses a centralized management to obtain a global view and the distribution of the bandwidth among peers. Centralized architectures are unscalable and expensive to maintain. In a semi-centralized architecture, all of the decision making is with either one peer or a small number of peers. Selfish behavior of peers can degrade the quality of service.

Chapter 3

Handling Flash Crowd in P2P Multi-Channel Live Video Streaming

The deployment of live video streaming applications has seen a large growth on the Internet [196]. P2P networks provide a robust, reliable and scalable message routing and delivery on top of a physical (underlay) network. Large-scale commercial P2P streaming networks such as Coolstreaming [200] and PPLive [166] offer many video channels to thousands of users, simultaneously. These are referred to as multi-channel P2P streaming networks [86]. In these types of networks, the peers can participate in more than one channel and switch between them. This chapter focuses on the following challenges: imbalance of resource usage and flash crowd.

The rest of this chapter is organized as follows: Section 3.1 describes the challenges of current architecture. Section 3.2 presents related work. The proposed framework is presented in Section 3.3 and 3.4.

3.1 Challenges

This section describes the following challenges:

- **Imbalance of Resource Usage:** One challenge with the use of multi-channel networks is an imbalance in resource (such as bandwidth) usage of different channels [86][40]. This implies that some channels have satisfactory streaming qualities since they have a surplus of resources, while other channels suffer from unsatisfactory streaming quality due to a lack of sufficient resources.

- **Flash Crowd:** A flash crowd occurs when a large number of peers attempt to join a popular channel. A flash crowd can result in high consumption of upload bandwidth of a P2P network and thus causes a decrease in Quality of Service (QoS). One result is the lag between choosing a video and actual playback (i.e., the startup delay is high).

- **Limited Available Bandwidth in P2P System:** In a P2P system, the available bandwidth is limited and constitutes a critical bottleneck for the deployment of large scale video streaming applications. There is a need for more effective use of existing bandwidth.

3.2 Related Work

This section presents previous work with regard to P2P multi-channel live video streaming networks.

3.2.1 Flash Crowd

Flash crowds occur frequently in P2P live streaming systems. The sudden arrival of numerous peers may consume the bandwidth of a P2P network, and decrease the QoS [114]. Existing methods for reducing the startup delay of a peer can be categorized into three types [114][40][189]. First, newcomers prefer to receive those chunks of video which have already been obtained by most other peers in the network. The reason for this is that this often increases the number of potential sources of the video chunk. However, if many peers already have the video chunk, then the video chunk may be removed from the buffer. Second, newcomers request video chunks sequentially, in order to ensure seamless playback. Therefore, the diversity among video chunks among peers that arrive in a short period of time is low. As a result, peers that are part of a flash crowd upload few video chunks among each other. Third, newcomers do not exchange their buffer-map [85], which indicates which video chunks currently exist in the player buffer of peer, until they obtain enough video chunks, to decrease the signalling overhead. Hence, before the peer announces its buffer-map, other peers do not know which chunks of video their neighbours have and cannot download from them.

From the scalability view of P2P live streaming networks, Chen et al. [186] showed that a P2P live streaming network with admission control under a flash crowd can increase its uploading capacity to support higher request rates. Liang et al. [112] used a branching process model to examine the service capacity of BitTorrent-like file sharing systems during flash crowds.

Load balancing problem in P2P networks is recognized as an important factor when ensuring that peers are not overloaded when flash crowd phenomena occur. Load balancing approaches depend on the type of P2P architecture [170]. For structured P2P system, Raoet et al. [114] proposed a scheme in which migration of a virtual server takes place between a set of heavy nodes and a set of lightly-loaded nodes. However, it has a serious drawback since it uses a number of directories to store and update the load information of the light nodes in the system, which consumes a large amount of resources in the P2P system. In unstructured P2Ps (e.g., Gnutella), random peer selection is commonly adopted. For example, Chawathe et al. [86] proposed a load balancing scheme based on the use of a random walk. It tries to increase the probability of visiting a lightly-loaded peer by organizing the overlay in such a way that a peer with high capacity has a large number of adjacent peers. There is a need for a global view of the system concerned with the distribution of the node capacities.

In file sharing, Garbacki et al. [186] proposed a load balancing scheme for hierarchical P2Ps, based on a list of super-peers called super-peer cache. In this scheme, a heavily-loaded super-peer can reduce the number of requests received from ordinary peers by reducing its priority in the super-peer cache held by each peer. This occurs when ordinary peers try to connect to the super-peers in the order indicated in the super-peer cache. However, a selfish behaviour of the super-peers significantly decrease the performance of the overall system, since it lacks an incentive mechanism.

3.2.2 Unbalanced Resource Distribution Among Channels

Multi-channel design techniques can be classified into three groups [184]; (1) In the Naive Bandwidth Allocation (NBA) approach [45], a peer subscribes only to its watched channels, and dedicates its upload bandwidth to them; (2) In the Passive Channel-aware Bandwidth Allocation (PCA) [181], a peer joins only to its watched channels, and optimally dedicates its upload bandwidth to these channels; (3) With the Active Channel-aware Bandwidth Allocation (ACA) approach, a peer joins not only to its watched channels, but also to several other channels as a supporter. The main idea behind ACA is to allow the channels with surplus upload bandwidth to help those with deficit upload bandwidth since the upload bandwidth is an important resource that greatly influences the quality of live video. In ACA, a peer optimally allocates its upload bandwidth to the watched channels and unwatched channels.

The View-Upload-Decoupling (VUD) [187] is the state-of-the-art framework for multi-channel live video streaming. VUD provides cross-channel resource sharing. In VUD, each peer is assigned to one or more channels as a supporter. Therefore, video distribution for unpopular channels is achieved by utilizing the bandwidth of supporters from other channels. However, it still suffers from upload bandwidth overhead and distribution swarm management cost [59]. Liang et al. [112] have proposed a partial decoupling strategy instead of complete decoupling for viewing and uploading of a peer in multiple channels. Some of the peers with rich upload bandwidth are assigned to help other channels.

None of the above work proposed a mechanism to handle the flash crowd phenomenon or provides incentives for peers to share their upload bandwidth [123][61].

3.2.3 Incentive Mechanism

Most incentive mechanisms proposed for P2P networks [199][146] are centralized. CFC proposed a decentralized incentive mechanism. This is done by allowing neighbouring nodes to disconnect from a peer that is not responsive to its requests. If node i categorizes node j as one that does not respond to requests or has not sent a message then node i disconnects from node j. If node j does not respond because it does not want to share its resources then node i will not respond to node j's requests. If a peer node is disconnected then it cannot receive video. This mechanism has elements of both encouragement and penalty schemes [146].

3.2.4 Network Coding

Nguyen et al. [131] used a network coding technique [62] to reduce duplicated storage. In Yang et al [192] the authors proposed a system, called Avalanche, to distribute large files by using network coding. Avalanche uses special sets of secure hash functions that support network coding operations. However, it requires very few computational resources. Li et al. [110] applied linear network coding to the evaluation of download finish times in a Peer-to-Peer network. They showed that coding can provide a robust optimal solution and better performance than routing in a dynamic network environment. Kehdi et al [100] and Wang et al [183] considered the implementation of network coding and P2P cooperative computing for wireless network and mobile devices. Wang et al [183] introduced the notion of regenerating codes, where a new peer only needs functions of the stored data from the surviving parent peers

to significantly decrease the repair bandwidth. In [182], the author proposed a P2P storage cloud for on-demand streaming.

3.3 Decentralized approach to handling flash crowds

This section presents the Coping Flash Crowd (CFC) framework [22], that represents a decentralized approach to handling flash crowds. This is handled with the run-time construction of a hybrid mesh-tree overlay structure at the release time of a video and adapted when needed (e.g., for peer-churn and, support load balancing). To support resource sharing an incentive mechanism is proposed that encourages peers to allocate their upload bandwidth to other channels. Since the quality of video at the peer side is related to its upload rate, it is very important to implement an incentive scheme for better streaming services. Without this mechanism, the flash crowd handling technique does not provide any advantages for reducing startup lag for peers.

3.3.1 Overlay Structure

The use of a single tree structure for delivering video requires a root node, which is the video source. A peer only has one parent and a video stream is sent through each path. One of the disadvantages of a single tree is that the upload bandwidth of a leaf node is not utilized. This is addressed in a multi-tree where the video source divides the stream into multiple sub-streams. There is a sub-tree for each sub-stream. A leaf peer can be an internal node of another tree. One major drawback of tree-based streaming systems is their vulnerability to peer churn. A peer departure will temporarily disrupt video delivery to all peers in the sub-tree rooted at the departed peer. In a mesh-based structure, peers dynamically connect to a subset of random peers in the system. A peer pulls video content from its neighbors who have already obtained the content. Since multiple neighbors are maintained at any given moment, mesh-based systems are robust to peer churns. However, different data packets may traverse different routes to a peer. The arrival of data packets is unpredictable and thus peers may observe video playback quality degradation that includes long startup delays, frequent playback freezes and low video bit rates.

CFC uses a multi-tree but each peer dynamically connects to a subset of peers to form a mesh. Each tree in the multi-tree is responsible for a sub-stream but the mesh allows a node to receive other sub-streams. This reduces the time to receive video content, and to handle the peer churn. This combination of a multi-tree and mesh provides a *pyramid-like* overlay structure.

The video source uses the following formula to assign a sub-stream: $i = GOP_j \bmod k$, where k is the number of sub-streams and GOP_j represents the j^{th} Group of Picture (GOP). A GOP is a group of successive pictures within a coded video stream. The number of sub-streams is equal to the height of the multi-tree. The height of the tree is not to exceed some threshold value, which in this work is the number of sub-streams. The tree height represents the maximum length from the video source to a leaf allowed. Longer lengths imply longer paths for video sub-streams to traverse, and hence have longer download times. The data structures,

types of peers and protocols needed to create and maintain the overlay are described in this section. These were designed to support load balancing and flash crowds.

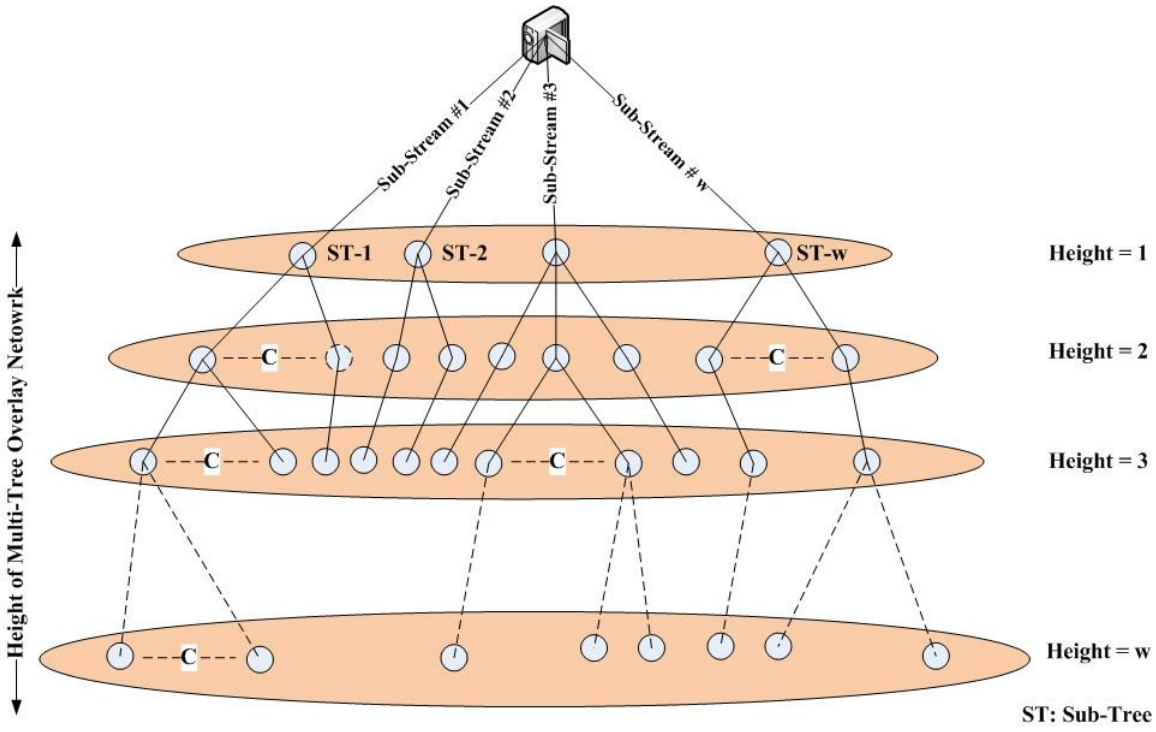

Figure 3.1: Multi-Tree Construction for Sub-Streaming

3.3.2 Bootstrap nodes

Bootstrapping is the process by which a new peer joins the overlay structure. The goal of the bootstrapping operation is to find a node that is already a member of the overlay for it to connect to. This requires the use of *stretch* information. For two nodes i and j, D_{ij} is the number of overlay hops between two nodes in the overlay network and the number of IP path hops is represented by d_{ij}. The stretch factor is defined as $ST_{ij} = \frac{D_{ij}}{d_{ij}}$. The stretch factor represents the overlay and underlay network mismatching (i.e., it represents the difference in the lengths of the shortest route between two nodes in the overlay and the shortest route between these nodes in the underlay network). For a node i and a set of nodes in set J the *average stretch value* can be calculated as seen in Equation 3.1:

$$Avg_{stretch} = \frac{\sum_{j=1}^{|J|} ST_{ij}}{\omega} \tag{3.1}$$

where $\omega = \frac{BW_{upload}}{Avg_{VBR}}$ and BW_{upload} is the average upload bandwidth the peers in set J, and Avg_{VBR} is the average video bit rate of nodes in J.

The *Global Delay Stretch* is the average stretch factor when i is the video source and J is the set of leaf nodes. This represents the average difference between play time of the video (which is streamed from the video source) and the time to play the video if it has to be transmitted to a leaf node in the tree. The *Local Delay Stretch* is the average stretch when i is a parent

and J is the set of its children. This value changes as children are added. A set of peers are designated as bootstrap peers. A bootstrap peer is the root of a sub-tree, which is associated with a sub-stream. Each bootstrap peer directly receives its sub-stream from the video source. Each bootstrap peer keeps for each node i in its subtree the global delay stretch and available upload time. A newly arrived peer sends a request to the video source for a list of bootstrap nodes. The new peer randomly selects a bootstrap node and requests a joining point (a peer node that it may become the child of). If A is the possible set of joining points then the joining point select is a node that with the addition of the newly arrived peers results in the smallest local delay stretch and the number of children associated with node i has not exceeded node i's limit. If there is no joining point the new peer will continue to contact bootstrap nodes until it finds a joining point. If no joining point is found, the new peer sends a message to the video source. The video source creates a new sub-stream.

The number of children that a node i may have is determined by the size of the sub-stream and the upload bandwidth of the node. The size of node i's sub-stream is denoted by S_i and is between between S_{min} and S_{max}, where S_{min} is equal to the size of each GOP and S_{max} is the length of the video stream. We assume that video has a variable bit rate which implies that S_{min} and S_{max} changes. If BW_i^u is the upload bandwidth of peer i in the multi-tree overlay network, the number of children for peer i in multi-tree is $C_i = \frac{BW_i^u}{S_i}$ where $\frac{BW_i^u}{S_{min}} \leq C_i \leq \frac{BW_i^u}{S_{max}}$. This calculation provides good QoS by not allowing an excessive number of peers to consume the upload bandwidth. If there are N peers let C be the $\min(C_i \mid 0 \leq i \leq N-1)$.

Assume that the number of sub-streams is h. The number of peers at level 2 (height 2) and level 3 (height 3) is $h \times C$ and $h \times C \times C$ respectively. The number of peers of peers at level i is $height_i = h \times C^i$. The minimal number (worse case) of peers in the tree is calculated as follows:

$$\sum_{i}^{h} height_i = h * \times (C + C^2 + ... + C^{h-1}) = h \times (\frac{1 - C^h}{1 - C}) \tag{3.2}$$

When a sub-stream is created each peer in the multi-tree is able to accept at least one more child. The reason for this is that the number of GOPs becomes less. This allows for accommodation of flash crowds. The child nodes of a bootstrap node are designated as *shadow* or *backup nodes*. The bootstrap peer for the new sub-stream is chosen from a set of backup peers. When a bootstrap node leaves, a shadow node is chosen to replace it. This mitigates the effects of peer churn.

3.3.3 Trackers

Tracker peers, which are a subset of peers, are used to discover possible mesh neighbours for newly arrived peers. To support a node joining a mesh requires the use of tracker peers. Each tracker maintains a node-handle list and is referred to as a Local Tracker (LT). A node-handle list maintains information for a set of peers. Each peer in the node-handle list is a distance of r from the local tracker in the underlay network. The maximum distance between two peers in the underlay network is assumed to be no more than $2r$. The information kept about a peer includes the peers IP address, availability time, and the bandwidth. When a new peer wants to join the network, it initiates a LT discovery procedure by sending a request to the video source that it wants to receive video from, a request for a list of LTs. Upon receiving the list of trackers from the video source, it sends a ping message to each tracker in the list within a distance r.

If the peer does not receive a message, it can then conclude that there are no available trackers within distance r. Therefore, the peer registers itself with the video source as a tracker. If the peer receives one or more responses to its ping messages, then the peer calculates the underlay distance between itself and the responding trackers. The requesting peer registers itself with the nearest tracker and requests a list of the active peers in the LTs node-handle list. Upon registration, the local tracker adds the registering peer to its node-handle list.

When a peer receives a list of neighbours it selects a set of neighbours for its mesh. This work uses physical network locality where choices are based on minimizing the traffic between the peer and selected neighbours. This is done by calculating the amount of traffic for the set of links, connecting the peer and a possible neighbour. The amount of traffic $T_{(t_a,t_b)}$ for sampling time (t_a, t_b) for each underlay link, i is calculated as follows:

$$T_{(t_a,t_b)} = \int_{t_a}^{t_b} link_i(t)\,\mathrm{d}t \tag{3.3}$$

By approximating the average number of links and routers in the underlay network between two peers it is possible to estimate the average load on those links. To find the average number of routers between two peers we need to find the shortest path between two random selected nodes in the graph. The work described in [49] [103] [142] shows that for any graph the shortest distance is approximately:

$$d = \frac{\ln[(N-1)(\hat{Z}_2 - \hat{Z}_1) + \hat{Z}_1{}^2] - \ln(\hat{Z}_1{}^2)}{\ln(\hat{Z}_2{}^2/\hat{Z}_1{}^2)} \tag{3.4}$$

where $\hat{Z}_1$ is the average number of k hop neighbours and N is the total number of vertices in graph which are considered as routers. $\hat{Z}_1$ and $\hat{Z}_2$ are defined as:

$$\hat{Z}_1 = \frac{[\sum_{x,y=1}^{N} A_{xy}]}{N} \tag{3.5}$$

$$\hat{Z}_2 = \frac{[\sum_{x,y=1,x\neq y}^{N} I_{\hat{A}}(x,y)]}{N} \tag{3.6}$$

In above equation the A is the routing adjacency matrix, and $\hat{A}$ is equal to A^2, and I is define as:

$$I_{\hat{A}}(x,y) = \begin{cases} 1, & if \hat{A}_{xy} > 0 \\ 0, & otherwise \end{cases} \tag{3.7}$$

The traffic between two peers A and B is based on the set of links connecting these two peers.

$$D = \sum_{i=A}^{B} link_i \tag{3.8}$$

Therefore the average traffic on the underlay network is $D \times T_{(t_a,t_b)}$. The lower the value of D the lower the amount of traffic in underlay network. This allows peers to choose a neighbour based on the physical locality. Furthermore, peers do not select neighbour peers randomly. Each peer in the network redirects requests to its neighbours based on the neighbour's response to requests. We defined four categories: (1) Those that have a high probability of a hit rate (meaning a response is returned); (2) those that have the minimum hit probability; (3) those that miss requests; and (4) those that the request message is not sent. To implement these scenarios, each peer categorized its neighbours based on the number of hits. Essentially, peers use their previous experience to direct their requests.

3.3.4 Buffer-map

A peer's buffer-map indicates which video chunks currently exist in the player buffer of a peer. Most buffer-maps consists of a zero or one representing the availability of a frame in the buffer of the peer. CFC proposes a new type of buffer-map structure to enhance video quality by reducing the end-to-end delay. This buffer-map includes four types of information: 1) Availability of the frames in the buffer; 2) Size of each frame in the buffer: 3) Estimation of response time; and 4) Average amount of free upload bandwidth.

By utilizing the buffer-map, peers estimate the time difference between the current time and the deadline of a chunk which we refer to as the *urgent factor*. A peer selects one of its neighbours to request a video chunk based on the urgent factor. We assume that a video chunk consists of l frames divided into c chunks, and f represents the frame rate of video (frames per second). Therefore, each chunk has the $\frac{c}{f}$ seconds of video. Moreover, the playback time of n-th chunk can be computed as $(n-1)\frac{l}{f}$. Let S_n be the size (measured in kilobytes) of n-th chunk, and T_{ij} be the transmission time between two peers i and j, and BW_i^u is the upload bandwidth of peer i. The n-th chunk should be delivered within the amount of time represented by $T_{ij} + \frac{S_n}{BW_i^u} \leq (n-1)\frac{l}{f}$. Equ. 3.9 shows that the average busy time for each neighbour of a peer.

$$T_{avg} = \frac{(1-\gamma)T_{avg} + \gamma T}{2} \tag{3.9}$$

where T_{avg} is the average response time of that neighbor, γ is a coefficient for controlling the effect of sudden changes of video rate. This means that when the video bit rate changes suddenly, the impact of video bit rate change should not affect the T_{avg} (refers to Equ. 3.9) [24]. The average response time is depended on the free upload bandwidth capacity of peer's neighbors. When video bit rate increases, it consumes more capacity of upload bandwidth of the peers to respond the video request message, and T is the necessary time to respond to a request by that neighbor in the last buffer-map exchange. We define T as follows:

$$T = \frac{S_i}{BW_i^u} + \frac{S_j}{BW_j^d} + T_{ij} \tag{3.10}$$

Where BW_i^d is the download bandwidth of peer i.

3.3.5 Incentive Mechanism

Since the quality of video at the peer side is related to its upload rate, it is important to implement an incentive mechanism for live video streaming. Free riders in the P2P networks are peers that only use services but provide little or nothing in return. Therefore, it is important to encourage the peers to act as supporters and allocate their upload bandwidth for other channels.

3.3.6 Performance Evaluation

For simulation, OMNeT++.4.1 [138] and the INET framework [169] is used to create a TCP/IP network. This framework implements UDP, IP, and Data Link Layer in OMNeT++. With OverSim [21], a framework in OMNeT++ for simulating P2P systems, the P2P overlay network is constructed. It utilizes the INET framework for simulating the underlay layers. In

general, OverSim prepares a reusable framework that has two main parts; (1) Overlay layer: for creating neighbor relation and constructing the mesh (tree or structured systems), and (2) Application layer.

In this simulation, the underlay topology is generated by using the Georgia Tech Internet Topology Model (GT-ITM) [34] tools for OMNeT++ v.4. This network is a decentralized and unstructured P2P network with 28 backbone routers, 748 access routers and 2000 peers. Peers in overlay network randomly select a router and connect to them by selecting random underlay link with a bandwidth between 128 Kbps to 2 Mbps and delays between 15 ms to 156 ms. In this simulation, video trace files are used from the Video Trace Library in [156] for streaming actual video. Table 3.1 shows the simulation parameters. In order to conduct a simulation that is more similar to the real world we generate the Constant Bit Rate (CBR) in the physical network. In order to load the network, it was imperative to set CBR background traffic to vary network load and enable us to study the proposed approach under different conditions. CBR traffic has been setup from various sources, with a 512 byte packet size. This background traffic operates during the entire duration of the simulations.

Table 3.1: Simulation Parameters

Simulation Parameters	Value
Video Codec	MPEG4
Video FPS	25
Number of Frame in GOP	12 Frames
Selected Trace File	Star Wars IV
Peer Upload Bandwidth	Random(128,2048) Kbps
Peer Download Bandwidth	Random(512,2048) Kbps
Average Video Bit Rate	512 Kbps
LifeTimeChurn	Weibull Distribution
Physical Link Delay	Random(9, 156) ms
Number of Channels	4

For peer churn, two different configurations are considered: (1) LifeTimeChurn. (2) NoChurn. In LifeTimeChurn configuration, when a peer is created, its life time is set randomly from the given probability function. When this life time is reached, the peer is removed from the network and a new peer will be generated. In Nochurn configuration, peers will be added until the target OverlayTerminalNum (the number of peers in the network) is reached and peers do not leave the network until the end of simulation. The focus on this work is not on scheduling algorithms and so a simple scheduling algorithm is used, similar to that proposed in [161]. The proposed framework is compared with View Upload Decoupling (VUD) which represents the state-of-the-art mechanism for multi-channel P2P systems.

End-to-End Delay

The average end-to-end delay is defined as the average time between transmission and arrival of data packets from source to destination. The end-to-end delay is impacted by the average length of paths from the video source to the peers and the network diameter.

In Fig. 3.2, the x-axis is the end-to-end delay and the y-axis represents values of the cumulative distribution function (CDF). A point (x,y) represents that y percentage of peers have an

average end-to-end delay that is less than or equal to x seconds. For example, in Fig. 3.2, 50% of the peers have an average end-to-end delay that is less than 7 seconds but in the VUD-like implementation, 50% of the peers have an average end-to-end delay of less than 10 seconds. This represents a significant difference in playing time and shows the approach to neighbour selection (selection is based on physical network locality). It also shows that the incentive mechanism used by CFC is effective but VUD is not.

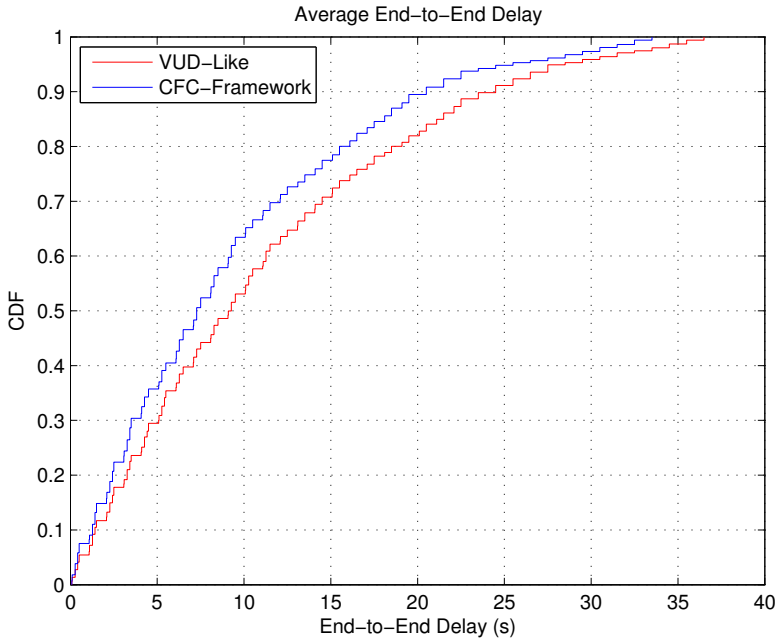

Figure 3.2: Average End-to-End Delay

Playback Delay

Playback delay (start-up latency) in live video streaming is the difference of video timestamps between starting transmission time in the source node and playing time in the destination peer. It shows the time taken to fill the player buffer of the peers by considering the startup buffering time. The value of this metric increases as the size of the network grows. Fig. 3.3 shows similar results to Fig. 3.2 and thus provides further evidence of the effectiveness of the mechanisms introduced for CFC that are not found in VUD.

Distortion

Distortion (or video packet miss ratio) is a performance metric which shows the percentage of video content that is lost compared to the original video. Equ. 3.11 shows how distortion is calculated. The distortion rate consists of two parts: (1) Packet loss due to loss in the underlay links; and (2) Loss from frame play timeout. Fig. 3.4 provides further evidence that the mechanisms introduced for CFC are effective.

$$Distortion = \frac{(Total\ Size\ of\ Received\ Frames) \times 100}{Total\ Size\ of\ Requested\ Frames} \tag{3.11}$$

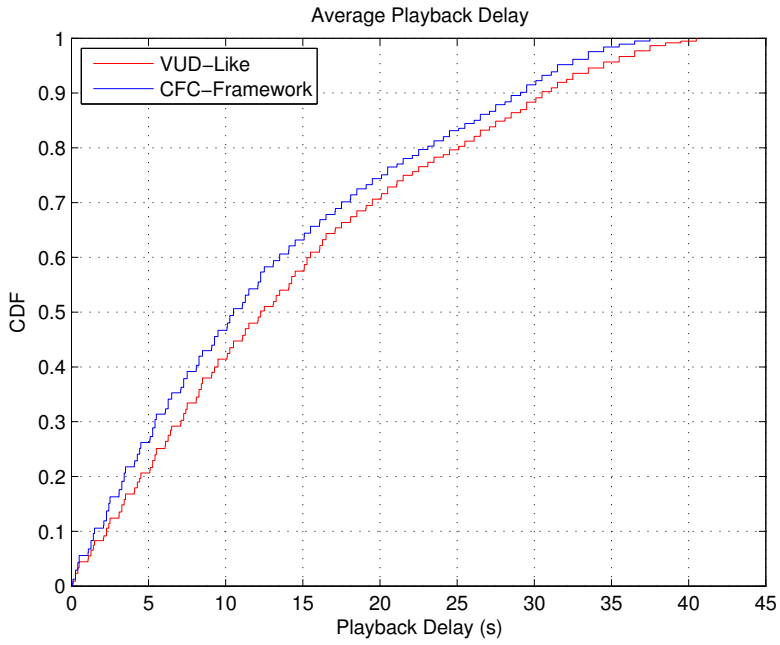

Figure 3.3: Average Playback Delay

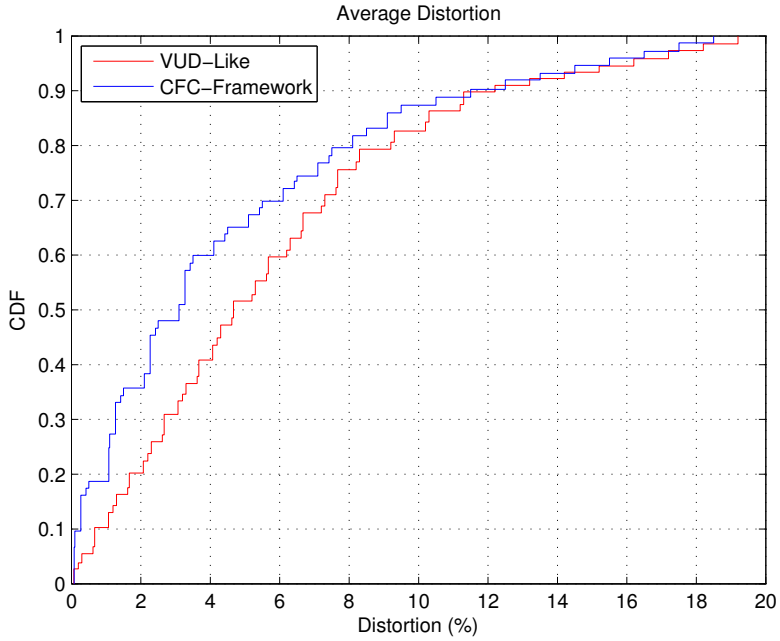

Figure 3.4: Average Distortion

Link Stress

Link stress [123] measures the number of replicated video packets that enter an autonomous system (AS) until all peers in that AS receive those packets. The lowest value is 1 which means

that only one packet enters into an AS, and all peers in that AS receive that packet. The maximum number of link stress is N which is the number of peers in the AS. As Fig. 3.5 shows, redundant traffic can be safely reduced by considering the underlay hop count in neighbour selection. In CFC, peers select their neighbours by using dual-mode locality awareness, and implicitly considering the underlay distance between themselves and other peers in the network. As a result, they establish the connection with those peers with shorter distance in the underlay. It is known that peers in the same AS have shorter distances in underlay network. Therefore, neighbour selection inside the AS, instead of across AS, can reduce the link stress. It is noticeable that time is a critical factor for live video streaming. Therefore, video frames should be delivered to the destination before their playback time.

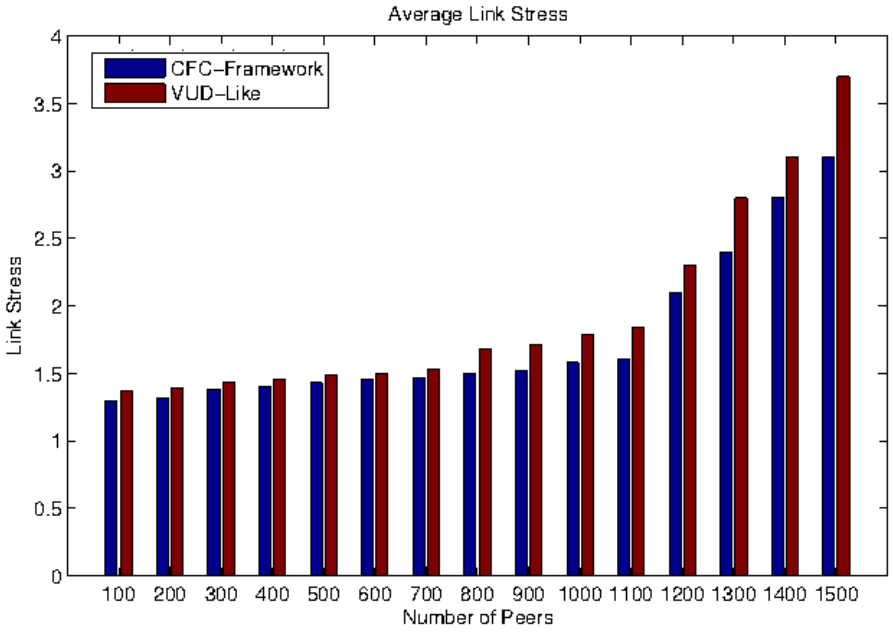

Figure 3.5: Average Link Stress

Stretch

A lower stretch results in a better query time and reduces unnecessary bandwidth consumption [123]. Fig. 3.6 shows that the CFC reduces the stretch factor in the P2P network since it considers the stretch factor in selecting is neighbours for the mesh. This allows for a reduction in both the number of underlay and overlay hop counts.

3.4 Reducing Bandwidth Consumption

Extended Coping Flash Crowd (ECFC) integrates network coding technology with sub-streaming to reduce bandwidth consumption [132]. The evaluation consists of extensive simulation studies that compare the proposed framework with View-Upload-Decoupling (VUD). VUD is commonly used as a comparison for multi-channel systems. Many mechanisms have been proposed

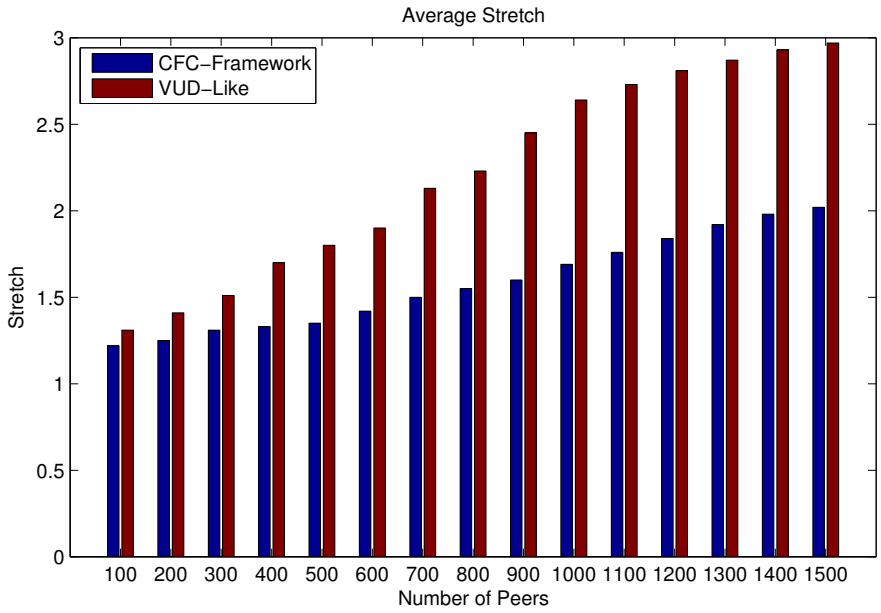

Figure 3.6: Average Stretch

in the literature and used in P2P streaming networks (e.g., [186][187][188][112]). This work includes handling flash crowds but most of the work assumes the use of a central management node. To see the problems with the use of a central management node, consider a scenario such as a hockey or soccer game where fans want to share their video. In this scenario the identification of a single management node that is persistent through the game may be difficult to identify. Load balancing techniques are often inadequate since peers are not always motivated to act as a supporter for other channels (intra-channel and cross-channel bandwidth allocation). To more efficiently use bandwidth, network coding was introduced [196].

3.4.1 Live Video Deployment Using Network Coding

Assume that a peer wants to send a stream of video to its neighbours. To increase the throughput, the peer first divides a video stream into n different chunks. The peers can then exchange their chunks among their neighbours. Since a peer downloads pieces of the video from its neighbours simultaneously, the time for a peer to recover all n chunks of the packets is potentially much shorter than that of downloading the file from only a single peer. Figure 3.7 shows the concept of network coding [132] in the framework.

Figure 3.8 shows how each peer in the framework utilizes the power of network coding technology to push sub-streams which are in its buffer to its neighbours. S_1, S_2, S_3, . . ., S_n are the input sub-streams to the peer's buffer. In this case, by using network coding technique each peer [132] encodes output sub-streams as a linear combination of the input sub-streams. In particular, assume that a_i, and b_i are new packets by linearly combining 4 chunks found in buffer of peers A and B respectively. Therefore, $a_i = \sum_{j=1}^{4} f_{ij}^{a} c_j$, $b_i = \sum_{i=2}^{5} f_{ij}^{b} c_j$, where f_{ij}^{a} and f_{ij}^{b} are random elements belonging to a finite field FQ [131], and $c_1, c_2, c_3, c_4, ...$ are the video chunks in the peers buffer-map of A and B. Assume that peers A and B are the neighbours

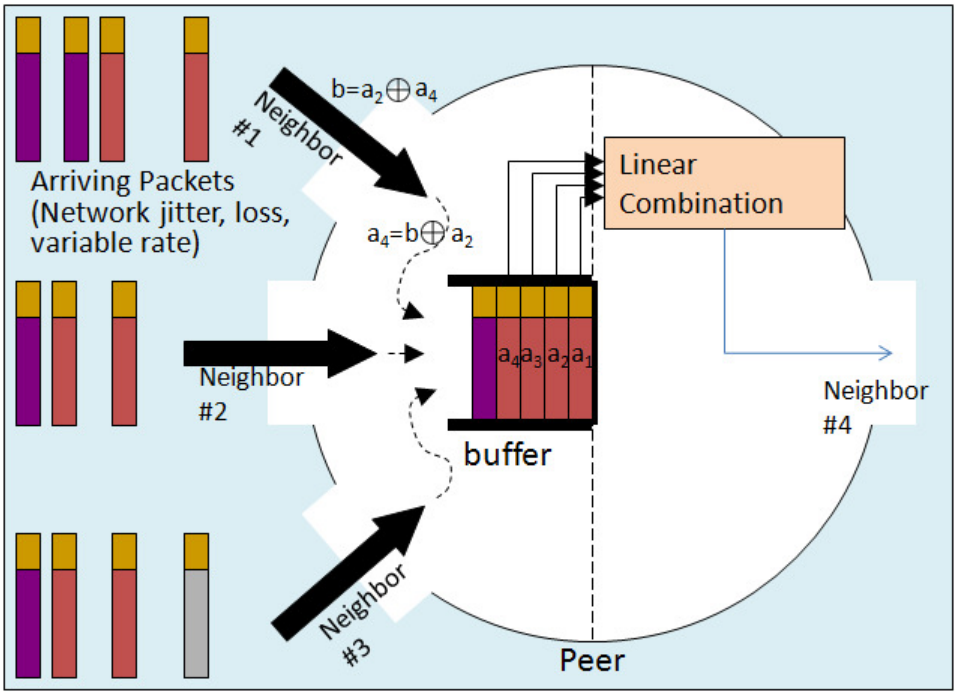

Figure 3.7: General Network Coding Technology [43]

of peer C. Peer C downloads $a_1 = f^a_{11}c_1 + f^a_{12}c_2 + f^a_{13}c_3$ and $a_2 = f^A_{21}c_1 + f^a_{22}c_2 + f^a_{23}c_3$ from A and $b_1 = f^a_{12}c_2 + f^a 13c_3 + f^a 14c_4$ and $b_2 = f^b_{22}c_2 + f^b_{23}c_3 + f^b_{24}c_4$ from B. Peer C will be able to reconstruct c_1, c_2, c_3, c_4 if f^a_{ij} and f^b_{ij} are known. Also, information about f^a_{ij} and f^b_{ij} can be included in the data packets. The number of bits required to specify f^a_{ij} and $f^b_{ij}$are $nlog(q)$ where n is the number of original packets while q is the size of the finite field. If $m >> n$ then these bits are negligible. Therefore, for most practical purposes, this network coding scheme can speed up the download time. However, in this type of network coding some of the packets received at a peer may be duplicate, and thus resulting in wasteful bandwidth.

Assume that the overlay network is a graph of vertexes and edges. Each vertex in this graph is a representation of a peer in the network and each edge represents the connection between neighbours. If node A has c incoming edges $\{e_1, e_2, .., e_c\}$ and one of the outgoing edges is e, then the edge function of edge e in a linear network coding scheme can be denoted by a vector $V_{ei} = \{f^{ei}_{ij}\}$. The message on edge e is generated by the linear function $M_e = f^{e1}_{1j}M_{e1} + f^{e2}_{2j}M_{e2} + f^{e3}_{3j}M_{e3} + ... + f^{ec}_{cj}M_{ec}$, where M_{ei} represents the message transmitted on edge e_i [47] [182].

In this framework, if a source has capacity k, the source can transmit sub-streams k simultaneously. If the k sub-streams are represented by a k dimensional vector M, the messages transmitted over edge e can be represented by the product of vector M and another k dimensional vector $V^{'}_{e}$. Vector $V^{'}_{e}$ is called the edge vector of edge e. Therefore, the source generates the sub-stream and uses network coding to push or (multicast) the sub-stream. The construction of the multi-tree is discussed in great detail in [22].

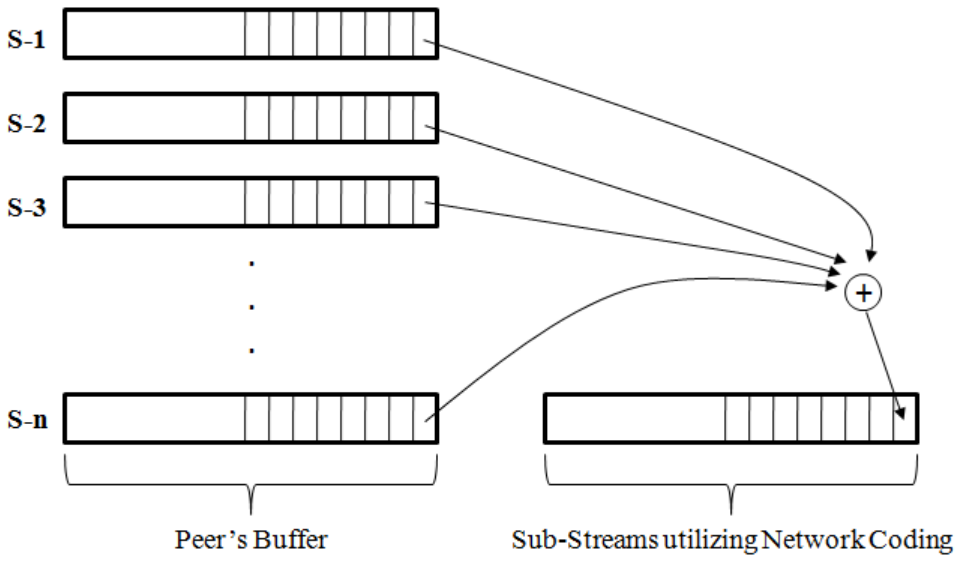

Figure 3.8: Integration of the Sub-streaming and Network Coding Technology

3.4.2 Performance Evaluation for ECFC

This section discusses the simulation results after incorporating network coding.

End-to-End Delay

In Fig. 3.9, the x-axis represents the average end-to-end delay and the y-axis represents values of the cumulative distribution function (CDF). A point (x,y) represents that y percentage of peers that have an average end-to-end delay that is less than or equal to x seconds. For example, Fig. 3.9 shows that 50% of the peers have an average end-to-end delay that is less than 7 seconds but in the VUD-like implemention we see that 50% of the peers have an average end-to-end delay of less than 10 seconds. This represents a significant difference in playing time. This shows that the approach to neighbour selection by CFC is effective compared to VUD. Fig. 3.9 shows a comparison of the end-to-end delay for CFC and ECFC. In ECFC peers try to establish a connection in the overlay tree as close as possible to the video source. To do so, they need to dedicate more resources to the network. This means that if any peer supports more peers in the network by uploading more video content, it can establish a connection to the higher levels of the tree. Accordingly, it will obtain the video packets in less time.

The comparison of CFC and ECFC shows the effectiveness of network coding in reducing the end-to-end delay. The network coding method combines received packets using a simple XoR operation as depicted in Fig 3.10(b). Peer $P3$ performs one less transmission using network coding which results in higher network throughput. Peer $P3$ in Fig 3.10.a needs to send both packets "b_1" and "b_2", but in Fig 3.10.b, by using a network coding, peer $P3$ only sends one packet (e.g,. $b_1 \oplus b_2$). Accordingly, peers $P5$ and $P6$ receive requested packets "b_1" and "b_2" with a lower end-to-end delay. The explanation is that the network coding reduces the number of relayed packets compared with the absence of network coding.

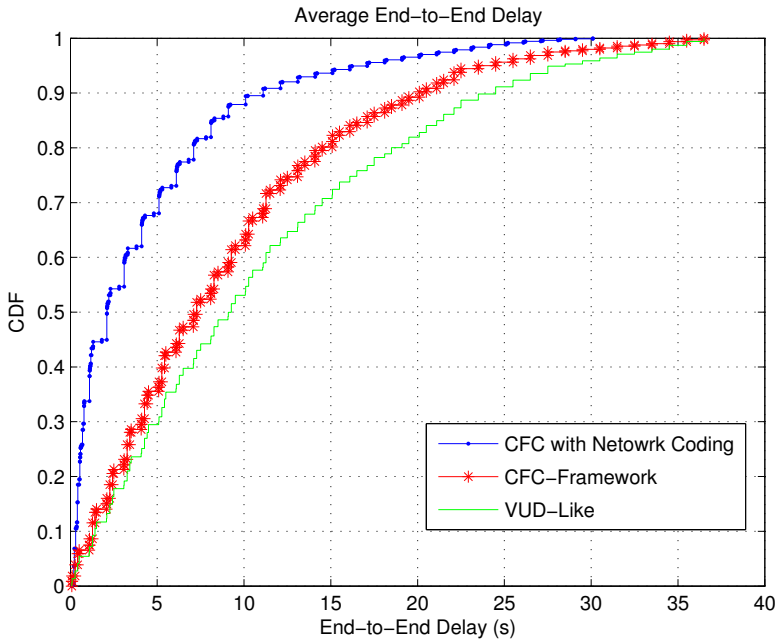

Figure 3.9: Average End-to-End Delay

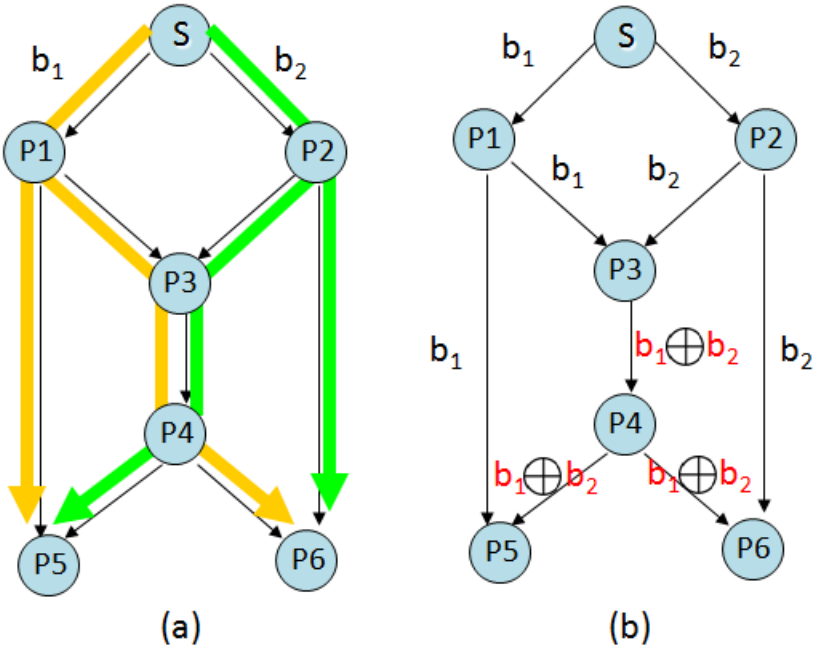

Figure 3.10: Packet forwarding without (a) and with network coding (b)

Average Playback Delay

Fig. 3.11 shows similar results as Fig. 3.9 and thus provides further evidence of the effectiveness of the mechanisms introduced for ECFC that are not found in VUD.

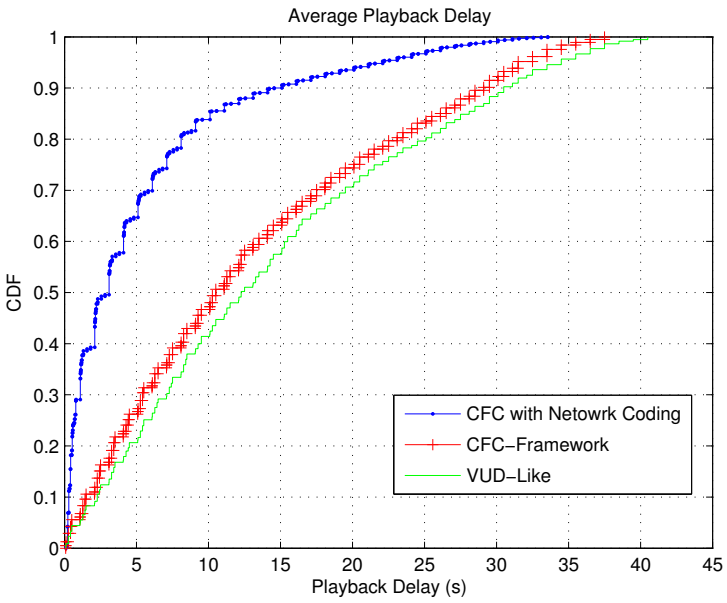

Figure 3.11: Average Playback Delay

Distortion

The distortion rate is impacted by the following: (1) Packet loss due to the loss in the underlay links; and (2) Loss from frame play timeout.

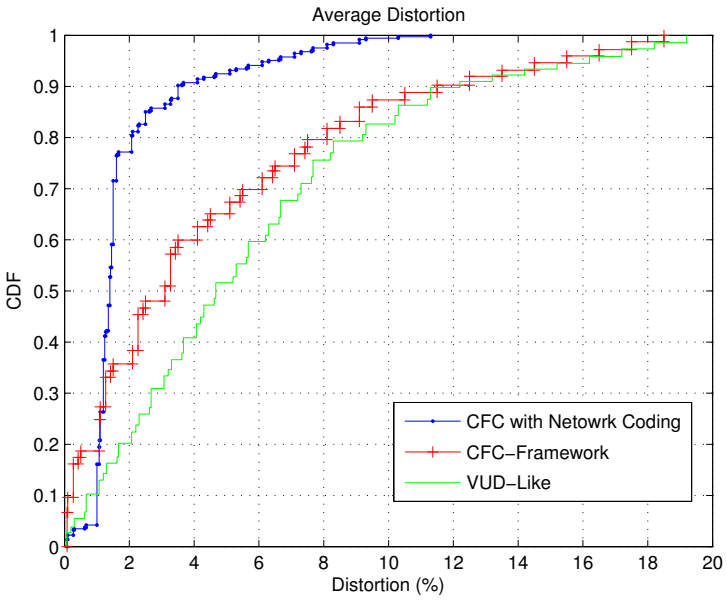

Figure 3.12: Average Distortion

3.5 Conclusion

This section summarizes the conclusions for CFC and EFC.

CFC is the first multi-channel P2P live video streaming architecture that considers mechanisms for coping with the flash crowd phenomena alongside the load balancing, incentive mechanisms and traffic localization. Moreover, CFC proposed an incentive mechanism that encourages peers to share their upload bandwidth between different channels. The performance evaluation results demonstrated that CFC can decrease the redundant traffic between ASs. In CFC, even though peers do not have rich resources, upload and download bandwidth could also contribute to the system. It can cope the flash crowd phenomena in P2P Network, and reduce the start-up delay because in live video stream networks, a new comer expected to watch a video immediately. Some of the critical factors that are considered in CFC design that help mitigate the flash crowd phenomena include: (i) underlay and overlay load balancing; (ii) incentive mechanism; (iii) traffic localization.

ECFC is the first multi-channel P2P live video streaming architecture that considers mechanisms for coping with the flash crowd phenomena alongside the load balancing, traffic localization and network coding. The performance of the framework was evaluated via extensive simulations and it was compared to the VUD approach. The results demonstrated that the redundant traffic between ASs decreases significantly. This provides strong evidence of the effectiveness of networking coding.

Chapter 4

MC-SkyNet: Mobile-Cloud Dynamic Partitioning for Mobile Cloud Applications

Mobile devices have limited resources including short battery life, storage capacity and processor performance. This limits the applications that can run on it. A mobile application can be partitioned so that some parts of the application runs on a cloud. This works well for applications with relatively little data to be transferred and that do not have a high level of interaction with the user. High latency is a challenge with applications that have large amounts of data to be transferred with a high level interactiveness. A cloudlet is a resource-rich computer or cluster of computers that is connected to the Internet and is available for use by nearby mobile devices. A mobile application can be partitioned so that part of it runs on the cloudlet. This work presents the MC-Skynet framework which introduces fine-grained offloading approach and support for runtime and dynamic partitioning of a mobile application. This is different from previous approaches, in that MC-Skynet does not only provides dynamic partitioning and offloading, but is also adaptive to the changes of the state of a cloudlet. It does this by introducing a cloudlet mesh network and self learning decision making module to estimate the offloading cost.

4.1 Introduction

Mobile device applications either entirely run on mobile devices or computation is split between the mobile device and a remote service. The remote service provides a well-defined API that can be used by mobile device applications (e.g., weather applications can use a remote service that collects weather data that becomes available through a well-defined API). The remote service may be hosted on a cloud. Regardless of the network distance between the cloud infrastructure and the mobile device, the use of a remote service is well suited for mobile device applications with relatively little data to be transferred. However, long network distances between mobile devices and remote services makes this approach unsuitable for applications that require larger amounts of data to be transferred and/or have a high level of interaction with the user. This includes mobile video communications (e.g., Skype, Face-Time, Google-Hangout),

gaming applications that require sophisticated rendering, traffic applications and cloud media analytics that can be used to offer more personalized services. Long network distances result in high latency that makes it difficult to support real-time and interactive applications. Latency can be reduced with the use of a *cloudlet* [164]. A cloudlet is a resource-rich computer or cluster of computers that is connected to the Internet and is available for use by nearby mobile devices [154]. Compared to the cloud, the cloudlet would be closer to the access point used by mobile devices to connect to the backbone.

There are several challenges with making effective use of cloudlets: (i) The nearest cloudlet in proximity to a mobile device may not have sufficient available computing capacity. In this MC-Skynet we propose a *cloudlet mesh*. A cloudlet mesh consists of multiple cloudlets that are dispersed through the Internet infrastructure and can communicate with each other; (ii) Often with mobile device applications some of the required computation takes place on the remote service. The part of computation that takes place on the mobile device at the remote service is static. The same partitioning strategy is not always suitable for all network conditions and inputs. For example, when considering a speech recognition application, performance depends on the size of the input and the type of connectivity to the backbone. If the connectivity to the backbone is through WiFi then more data can be transferred than if connectivity is through a cellular network. There is a need for a dynamic offloading strategy that considers the type of network connectivity and the amount of data being transferred. The amount of data to be transferred is not always known since this depends on the user.

MC-Skynet presents a dynamic partitioning strategy that considers network conditions and the amount of data being transferred, shows how a cloudlet mesh can be used, and presents performance results that illustrate the advantages of a dynamic partitioning strategy and a cloudlet mesh.

The rest of this chapter is organized as follows. In Section 4.2, we survey the literature related to the principal themes of this MC-Skynet framework. The proposed framework is presented in Section 4.3 . In Section 4.4, the performance evaluation of the proposed framework is provided. Finally, Section 4.5 summarizes the results and presents the conclusion and gives a description of future work respectively.

4.2 Related Work

This section describes some of the representative work in the literature on dynamic partitioning and offloading of applications [164] [105].

4.2.1 Partitioning

Application partitioning refers to dividing an application such that part of the application runs on the mobile device and the other part runs on a remote server. Partitioning decisions made during development refers to as *static partitioning*. If the partitioning decision is made during execution it is referred to as *dynamic partitioning*. Static partitioning has low overhead in that no time is used in determining a partition. However, static partitioning is not able to guarantee the best application partition for all possible execution environments. Dynamic partitioning makes decisions based on run-time conditions. However, this imposes additional overhead

since it requires run-time profiling [53] [46]. Representative work in dynamic partitioning is described in the rest of this section.

Cuervo et al [53] propose, MAUI, which allows developers to annotate the methods of an application that can be offloaded to a remote server. MAUI uses on-line profiling to determine if future invocations of annotated methods should be offloaded. In making decisions about offloading MAUI considers the number of CPU cycles that would be reduced if the method is offloaded (this directly relates to CPU consumption) and the amount of state information to be transferred to the remote server. The methods that can be offloaded are determined during development but the decision for offloading is at run-time. CloneCloud [46] is similar to Maui but developers are not expected to annotate their programs.

Ra et al [139] propose, Odessa, a dynamic partitioning-offloading tool for interactive applications. Odessa models applications as data flow graphs. Each node represents computation and an edge between nodes represents a flow of data. At run-time an application profiler determines any performance bottlenecks at any of the nodes or edges. One possible action is to offload a node that is a bottleneck to a remote server. During run-time it is possible that several nodes are offloaded.

Qi et al [193] present a genetic algorithm for application partitioning. They argue that the genetic algorithm is more likely to converge to the global optimal partition. However, the complexity of genetic algorithm is very high and it takes a long time to come up with optimum partitions.

Our work differs in that we consider the communication costs between nodes. Our work is able to calculate communication costs based on run-time behaviour.

4.2.2 Offloading

Dynamic partitioning requires the offloading of tasks to a remote server. Remote execution incurs some overhead e.g., transfer of input data over the network to the remote server and the transfer of the results from the remote server to the mobile service. To minimize the overhead some of the existing work (e.g., [105] [17] [104] [145]) suggest that tasks offloaded should represent large amounts of execution times. This is not always advantageous because of communication costs associated with the transfer of data between the mobile device and the remote server. Many methods have been proposed in the literature for making offloading decisions (e.g., [106] [95]). Typically the developers implicitly assume that the same copy of application is also installed in the remote server.

On-line profiling and history-based approaches can be used for predicting the offloading cost based on previous behaviour. Attributes used to determine costs include speed-up factor, power consumption and available bandwidth. The current load of both mobile device and the remote server effect the cost of offloading; however, none of the previous efforts considered this in cost estimation [158].

An example of using on-line profiling for offloading decisions is Spectra [67]. Spectra focusses on reducing latency and energy consumption by monitoring energy consumption of local and remote execution. Therefore, when the input data of a task changes, Spectras estimations become inaccurate and it requires significant effort from developers. Chroma [18] tries to improve Spectra by reducing the burden on developers. Chroma, like Spectra, uses a history-based approach to predict future resource demands. It should be noticed that Spectra

and Chroma both assume that the application is installed on the surrogates and there is no need to ship the application code.

Slingshot [172] addresses the latency issue by first trying to use a server accessible by LAN. If not possible it uses the Internet to connect to a remote service. The higher delay and lower bandwidth of the Internet slows task offloading to the remote server.

4.3 MC-Skynet Overview

The architectural framework for MC-Skynet is graphically depicted in Figure 4.1. MC-Skynet provides a framework for making decisions on whether to offload and which computational components are to be offloaded.

MC-Skynet consists of the following components: An application partitioning module that computes the set of application components to be offloaded. This component constructs a graph where each vertex represents application components that are tightly coupled. The prediction module is used to predict the amount of data to be transferred between two application components based on information on the amount of data that has already been transferred The graphs and the predictions from the prediction module are used by the decision making module to determine the applications to be offloaded and whether offloading should occur. This is described in more detail in Sections 4.3.1 and 4.3.2. Section 4.3.3 discusses how a mesh of cloudlets is used.

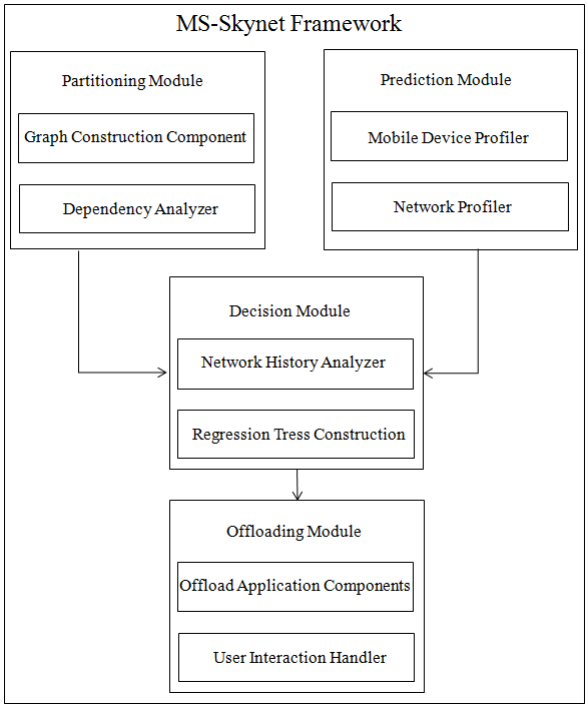

Figure 4.1: Framework

4.3.1 Partitioning

This section describes a dynamic partitioning approach that is used to determine the set of application components that are to be executed on the cloud. This allows the set of application components that are to be executed on the cloud to change while the application is executing in response to changes in the network connectivity to the backbone network e.g., a switch from WiFi to 3G. For determining the partitions of the applications the data flow between components which depends on the user and changes over time is considered. This is dealt with by periodically calculating the amount of data sent between application components.

Assume that M is the set of application components. The goal of partitioning is to find sets $P^{MD} = \{P_i^{MD} | i = 1, 2, 3, ..., k\}$ and $P^{RS} = \{P_i^{RS} | i = 1, 2, 3, ..., l\}$ such that $P^{MD} \cup P^{RS} = M$. The set of application components in P^{RS} is to be offloaded to a cloudlet. Assume that $G = (V, E)$ is an application graph where V is the set of application components i.e., $V{=}M$. Each edge between vertices, v_i and v_j, is associated with a weight, w_{ij}. The weights are periodically adjusted during run-time in fixed intervals of time and represent communication cost.

Weight calculation is based on the predicted amount of data to be transferred. At time $t + 1$ for each pair of vertices, v_i and v_j, the predicted amount of data to be transferred, ${}^*D_{t+1}^{ij}$ is calculated using Equ. 4.1. The calculation is based on the amount of data transferred in time slots t and $t - 1$ and is represented by D_t^{ij} and D_{t-1}^{ij} respectively [122][163].

$$ {}^*D_{t+1}^{ij} = \frac{D_t^{ij} \times \omega^2 + D_{t-1}^{ij} \times (1 - \omega)^2}{2} \tag{4.1} $$

There may be large fluctuations in the data transferred. We use a controlling coefficient, ω, to reduce the impact of sudden changes. This controlling coefficient may be different for different applications. Applications that are highly time sensitive may have a higher value for ω.

The weight (communication cost) for each pair of application components, i and j, is calculated using Equation 4.2.

$$ w_{ij} = \frac{{}^*D_{t+1}^{ij}}{B_{t+1}} \tag{4.2} $$

where B_{t+1} represents the average bandwidth in the time period between t and $t + 1$ and and w_{ij} is the calculated communication cost. An estimate of the average bandwidth at time B_0 is based on a technique described in [90] which makes use of a ping message. MC-Skynet determines the value of B_0 at the start of the execution of the application. Values of B_t where t is greater than zero is calculated as follows: For each data transfer between the mobile device and the cloudlet, timestamps are associated with the start of the transfer from the sender and the end of the transfer of the data. The mobile device uses the difference in time to determine how long it took for the data to arrive. Since the data size is known and the amount of time it took to transfer the data is also known, it is now possible to estimate the available bandwidth for that data transfer. The value of B_{t+1} is the average of the estimated bandwidth for data transfers that took place between time t and $t + 1$.

The partitioning of the set V is based on the edge weights. The weights are used to partition the graph vertices into disjoint subsets where each subset represent a group of application components that are to be considered as a single unit. Application components in these units are either all offloaded or will all remain on the mobile device. Each of the subsets represents

a multi-node. Let P_{ij} represents the set of edges on the path from v_i and v_j. The distance of locality, L_{ij}, between two vertices v_i and v_j is defined as the sum of the edge weights of the edges found in P_{ij}. If the value of L_{ij} is below some threshold value then the vertices v_i and v_j are put in the same set. If any of the vertices in a disjoint set directly depends on a device driver then that set is said to be device dependent. These sets are used to create a new graph, V', where each vertex corresponds to a disjoint set. For any two vertices V_i and V_j an edge exists in E' if at least one node in V_i has an edge with a node in V_j in E. The new graph constructed is represented as $G' = (V', E')$.

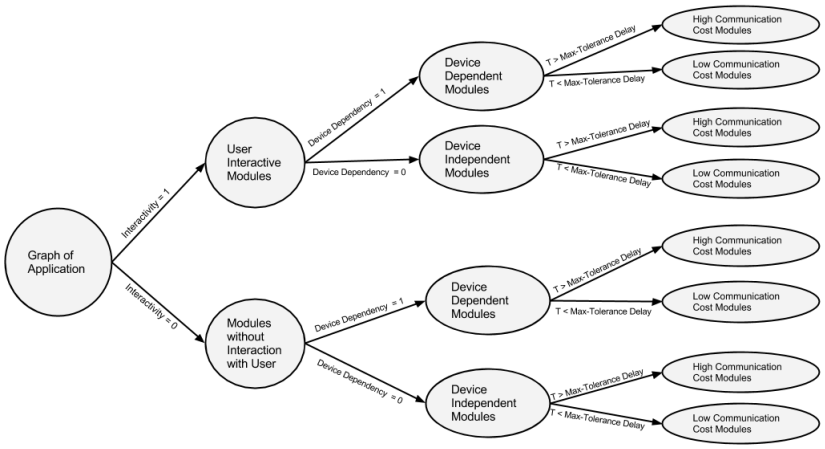

Figure 4.2: Regression Tree Structure

The sets, P^{MD} and P^{RS}, are determined using a regression tree that is applied to G'. The regression tree is a decision tree that uses a tree-like graph or model of decisions and their possible consequences, including chance event outcomes. It uses a decision tree as a predictive model which maps observations about a vertex in G to conclusions about the vertex's target value. In these tree structures, leaves represent class labels and branches represent conjunctions of features that lead to those class labels.

Figure 4.2 shows the regression tree which is constructed in the MC-Skynet framework. For different types of network like 3G, 4G, Wi-Fi, and etc, different regression trees are constructed. The regression tree partitions the vertices in G' into two sets: those application components that interact with the user and those that do not. These sets are further partitioned based on device dependencies. The last partition is based on the tolerance delay of the application.

4.3.2 Offloading

Section 4.3.1 describes what is involved in making a decision regarding *what* is to be offloaded. This section focuses on what is involved in making the decision on *whether* to offload or not. The analysis described in this section is carried out when there is a change in network connectivity

We assume that a mobile device is associated with a cloudlet. The cloudlet is chosen based on its proximity to the mobile device. More information on how a cloudlet is selected is found in [101]. We assume that the execution time on the remote site (e.g., cloud or cloudlet) can be

approximated by $ET_m \times \lambda$ where ET_m and λ represent the execution time on the mobile device and speed up factor of running the execution time on the cloudlet respectively[163].

The time it takes a result to return to a device from the part of the mobile application that has been offloaded to a remote site (cloudlet) is denoted by ET_c. An approximation of ET_c is represented by Equ. 4.3[163].

$$ET_c \approx ET_m \times \lambda + \frac{V_c + V_r + V_d}{BW_i} \tag{4.3}$$

where V_c is the size of the offloaded set (P_{RS}), V_d represents the size of the data which is transferred between application partitions P^{MD} and P^{RS} over the network, and V_r is the size of result in the previous communication between the application components in P^{MD} and P^{RS}. At time $t + 1$, for any two components, v_i and v_j, where v_i is in P_{RS} and v_j is in P_{MD}, the value of $*D^{ij}_{t+1}$ is calculated. The sum of of these values is the value of V_r. We assume that BW_i is the bandwidth available for a network of type i. Between a mobile device and a cloudlet the different types of network include WiFi and cellular. As can been seen, Equ. 4.3 considers both computation cost as well as communication costs.

If $ET_m < \frac{V_c}{BW_l}$, offloading does not improve the performance. The mobile device will decide to offload if $ET_m \gg ET_c$.

Let BW_l represent the end-to-end bandwidth which is large enough for offloading. If the estimated available bandwidth is less than BW_l then offloading is considered worthwhile.

The value of BW_l should have the property specified in Equation 4.4. Essentially BW_l should be sufficiently large enough that the transmission of V_c (the size of the tasks to be offloaded) should be significantly less than the estimated difference between execution all tasks on the mobile device and execution of some of the tasks in the cloud.

$$BW_l \gg \frac{V_c}{ET_m - ET_m \times \lambda} \tag{4.4}$$

4.3.3 Use of a Cloudlet Mesh

The discussion so far assumes a single cloudlet. A cloudlet may have many mobile devices associated with it. A cloudlet could become overloaded. We take advantage of nearby cloudlets which create a mesh network that uses Peer-to-Peer (P2P) protocols. The use of a mesh of cloudlets reduces the cost of utilizing the cloud, avoids the long latency introduced by wide-area networks for accessing the cloud, and localizes the Internet traffic and thus reduces the cost for the Internet Service Providers. By using the mesh of cloudlets we have a large amount of computational resources in the neighbourhood of end-users. This means that if the cloudlets to which the end-user is connected is busy or it does not have sufficient resources, it could redirect the request to its neighbours in the cloudlet mesh.

The introduction of a cloudlet mesh results in two types of offloading decisions. The first type of offloading occurs when a decision is made to offload application components to a cloudlet. The second type of offloading decisions occurs when a cloudlet becomes overloaded and requires the components of an application to be offloaded to another cloudlet. In the cloudlet mesh, the cloudlets exchange their availability of spares resources with each other periodically. Assume that cloudlet A wants to offload something to one of its neighbours. It

first finds the neighbour which has more available resources (assume find cloudlet B). It then sends a message to cloudlet B and reserves the resources of the cloudlet B for next time slot and after receiving an acknowledgement, it transmits the components to cloudlet B.

By utilizing the cloudlet mesh, mobile application service providers are be able to provide better services to mobile users by adapting their deployment services regarding with user mobility that might cause service degradation. Moreover, different types of networks such as 3G, Wi-Fi, and LTE have different feature and bandwidth. Also, user mobility causes network disconnection which causes more energy consumption and delay. Therefore, mobility of mobile devices should be taken into consideration for making offloading decision.

MC-Skynet either runs on the mobile devices or cloudlet. Since MC-Skynet needs to monitor the different parameters such as network bandwidth, resource consumption on the mobile devices, it needs to run on the mobile device. It also runs on cloudlets since cloudlets must also make offloading decisions. MC-Skynet dose not consider the battery and energy consumption for mobile devices into consideration. All the decision making regarding with partitioning and offloading takes place at the nearby ckoudlet.

4.4 Performance Evaluation

MC-Skynet proposes an estimation of communication and offloading costs as part of the decision making related to application partitioning and offloading. For measuring the offloading cost, MC-Skynet considers communication traffic, the size of data being transferred and execution time. The advantage of using a cloudlet mesh is that, it could localize the traffic and accordingly reduce Internet backbone traffic and the response time.

4.4.1 Simulation Setup

For simulation, OMNeT++.4.1 [138] and the INET framework [169] is used to create a TCP/IP network. This framework implements UDP, IP, and Data Link Layer in OMNeT++. With OverSim [21], a framework in OMNeT++ for simulating P2P overlay systems, we construct the P2P overlay network. It utilizes the INET framework for simulating the underlay layers. In our simulation, the underlay topology is generated by using Georgia Tech Internet Topology Model (GT-ITM) [34] tools for OMNeT++ v.4. In this simulation, the underlay network has 28 backbone routers and 748 access routers. In order to have a simulation which is more similar to the real world we generate the Variable Bit Rate (VBR) in the physical network. The VBR traffic has been setup from different sources to different destinations with packet sizes between 64 to 512 bytes. We simulate changes in connectivity by randomly changing the background traffic generated. Low connectivity is associated with high amounts of background traffic.

The simulation scenario is as follows. In this simulation there are muliple players in an on-line game. This on-line game has a computational part and a live video streaming part. For the computational part each player finds the matrix inverse of a matrix A. The matrix A is a buffer-map. Buffer-maps are exchanged among players periodically. Each player calculates A^{-1} and sends it to other players. After each player receives A^{-1}, the player computes A and then it is able to sends back the request message for receiving some parts of video. We compare different

scenarios: (1) players directly connect to a server inside the cloud; (2) players use a cloudlet mesh; and (3) each player connects to a cloudlet. Table 4.1 shows the simulation parameters.

Table 4.1: Simulation Parameters

Simulation Parameters	Value
Video Codec	MPEG4
Video FPS	25
Number of Frame in GOP	12 Frames
Selected Trace File	Star Wars IV
Mobile-user Upload Bandwidth	Random(128,2048) Kbps
Mobile-user Download Bandwidth	Random(512,2048) Kbps
Average Video Bit Rate	512 Kbps
Physical Link Delay	Random(9, 156) ms

In our simulation we design a processing unit for the mobile devices, cloudlets and the cloud. We model each processing unit as a $M/M/1$ queue. This model represents $Q_i = \{Q_i|1, 2, 3, ..., K\}$ which is a $M/M/1$ queue, where the first M represents the arrival rate, the second M represents the service rate and the 1 indicates that there is only one service unit in the system. Assume that $C_i = \{1, 2, 3, ..., n\}$ is the set of application components.

If the γ is the arrival rate and the μ is the service rate of resource R, then the throughput of r is $\rho = \frac{\gamma}{\mu}$. The expected queue length is $E[q_l] = \frac{\rho}{1-\rho}$ and the expected average total time (queuing time plus service time) is $E[T_c] = \frac{\rho}{\gamma(1-\rho)}$, where $E[T_c]$ is the total accomplish time. Also, the average waiting time is $E[T_w] = E[T_c] - \frac{1}{\mu}$ where $\frac{1}{\mu}$ is the service time. Thereby, we could measure the execution time of each application component on the mobile device or on the cloudlet or the cloud.

4.4.2 Evaluation

Response Time

The average response time is defined as the average time between transmission and arrival of data packets from source to destination. In fig. 4.3 the x-axis is the response time and the y-axis represents values of the cumulative distribution function (CDF). A point (x,y) represents that y percentage of peers have an average end-to-end delay that is less than or equal to x seconds. For example, in fig. 4.3 we see that 50% of the mobile users have an average response time that is less than 1.5 seconds. This does represent a significant difference in playing time. Accordingly, we conclude hat the approach to utilize the cloudlet mesh and localizing the computations is effective.

Average Computation Cost

The average computation cost is defined as the time which is needed to accomplish the computation part in the simulation. In the other words, it shows the average time for calculation the matrix inverse of the buffer-map. Fig. 4.4 shows the performance comparison of running an ap-

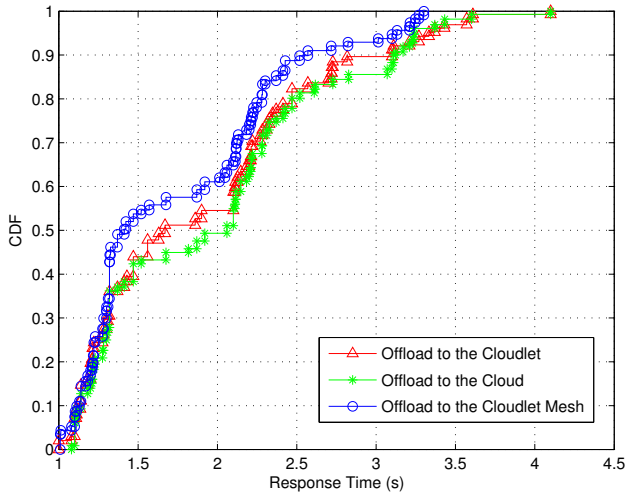

Figure 4.3: Average Response Time

plication stand-alone on the mobile device versus three different scenario for remote execution to servers that are successively further away.

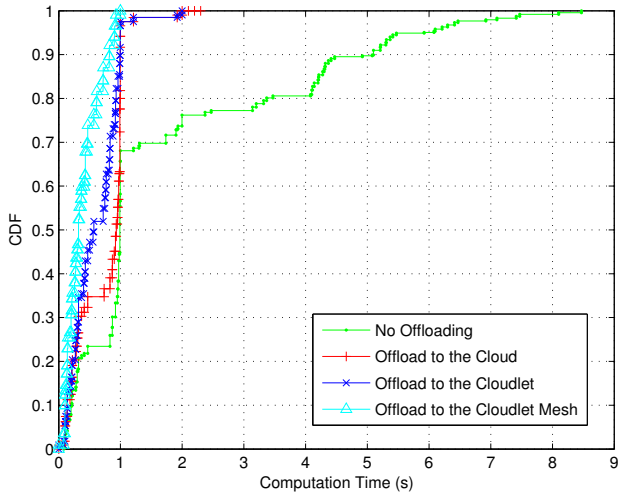

Figure 4.4: Average Execution Time

Distortion

Distortion (or video packet miss ratio) is a performance metric that represents the percentage of video content which is lost compared to the original video. Equ. 4.5 shows how we calculate distortion. The distortion rate consists of two parts: (1) Packet loss due to the loss in the underlay links; and (2) Loss from frame play time-out. Since the underlay network is the same for the all systems, the loss due to the physical link is the same for all algorithms in the

framework. Fig. 4.5 provides further evidence that the mechanisms introduced for MC-Skynet are effective.

$$Distortion = \frac{(Total\ Size\ of\ Received\ Frames) \times 100}{Total\ Size\ of\ Requested\ Frames} \tag{4.5}$$

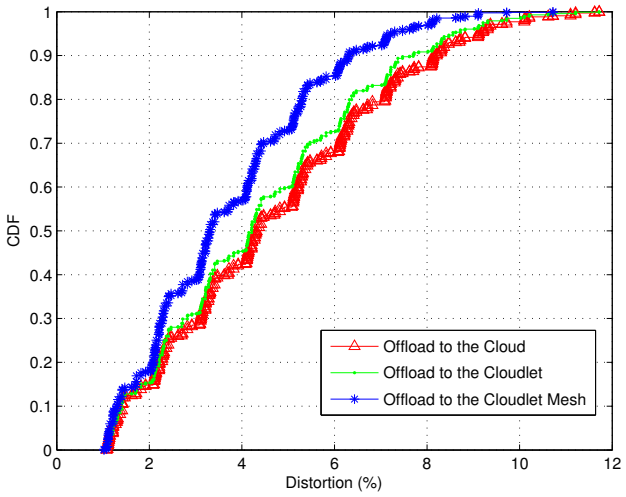

Figure 4.5: Average Distortion

Throughput

Assume that the C_{ij} is the communication cost (time). In the application graph it represents the weight on the edge. Let CP_i be the computation cost (time) which is representative of the weight in the vertex in the application graph. Therefore, the throughput is calculated as follows:

$$TP = \frac{1}{max\{max(CP_i), max(C_{ij})\}} \tag{4.6}$$

On the other hand, throughput means the number of data input data the data flow is able to process per second.

Estimating the Time Constraint

In the mobile devices side it is necessary to estimate the last possible time which result has to get back to the mobile device. Let x_i be the observed end-to-end bandwidth and the y_i is the predication end-to-end bandwidth and H is the number of observation and predication during the application execution. By calculating the Mean Square Error (MSE) we could compute the End-to-End Square Prediction Error (ESPE) during the executing time (refer to Equ.4.7).

$$ESPE = \frac{1}{Z}\sum_{i=1}^{H}(x_i - y_i)^2 \tag{4.7}$$

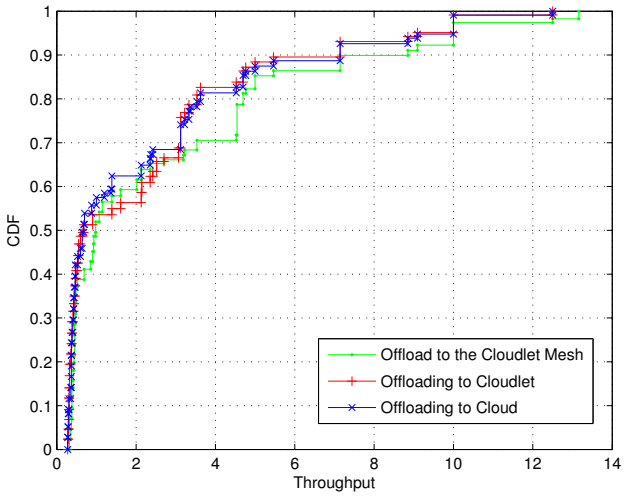

Figure 4.6: Average Throughput

By approximating the average number of links and routers in the underlay network between two points it is possible to estimate the average delay between those points. In MC-Skynet cloudlets are able to estimate the delay between each other in the clouldlet mesh network. To find the average number of routers between two cloudlets we need to find the shortest path between two random selected nodes in the graph. The work described in [49] [103] shows that for any graph the shortest distance is approximately:

$$d = \frac{\ln[(N-1)(\hat{Z}_2 - \hat{Z}_1) + \hat{Z}_1^{\,2}] - \ln(\hat{Z}_1^{\,2})}{\ln(\hat{Z}_2^{\,2}/\hat{Z}_1^{\,2})} \tag{4.8}$$

where $\hat{Z}_1$ is the average number of k hop neighbours and N is the total number of vertices in graph which are considered as routers. $\hat{Z}_1$ and $\hat{Z}_2$ are defined as:

$$\hat{Z}_1 = \frac{[\sum_{x,y=1}^{N} A_{xy}]}{N} \tag{4.9}$$

$$\hat{Z}_2 = \frac{[\sum_{x,y=1,x\neq y}^{N} I_{\hat{A}}(x,y)]}{N} \tag{4.10}$$

In above equation the A is the routing adjacency matrix, and $\hat{A}$ is equal to A^2, and I is define as:

$$I_{\hat{A}}(x,y) = \begin{cases} 1, & if \hat{A}_{xy} > 0 \\ 0, & otherwise \end{cases} \tag{4.11}$$

From what we discuss above we see that the cloudlet mesh is able to localize the traffic and as a result the communication time could be reduced. Moreover, cloudlet mesh by finding the path between two cloudlets inside the mesh which has a minimum hop distance (d in Equ. 3.4) is able to reduce the communication time.

4.5 Conclusion

MC-Skynet presents a dynamic partitioning strategy that considers network conditions and the amount of data being transferred, shows how a cloudlet mesh can be used, and presents performance results that illustrate the advantages of a dynamic offloading strategy and a cloudlet mesh.

Chapter 5

Related Work on Data Stream Processing

This chapter presents background information and a comprehensive description of work related to data streams, geo-referenced data streams and continuous queries.

This chapter is organized as follows: Section 5.1 discusses the related work on data stream processing and addresses the challenges. In Section 5.2, related work on management systems for geo-streaming applications is explained. Section 5.3 discusses query operator placement related work. Last, section 5.4, explains the gap introduced by the current architecture for data stream processing.

5.1 Data Streaming Processing

Data Stream Management Systems or data stream processing has received considerable research attention. This section describes data stream management systems and specific data stream platforms.

The key difference between a classical Database Management System (DBMS) and a Data Stream Management System (DSMS) is the data stream model [6]. A traditional database query executes once and returns tuples reflecting the current state of the tables. A tuple is defined as a data item embedded in a data stream and can be processed by a processing unit [55]. With data streams, data elements arrive on-line and stay only for a limited time period in memory [75]. Consequently, the DSMS has to handle the data elements before it runs out of memory. Also, the order in which the data elements arrive cannot be controlled by the system. Unlike one-time queries to regular databases, continuous queries continuously deliver results to an output stream. A continuous query is terminated either explicitly by the system or by a termination condition in the query. Since continuous streams may not terminate, intermediate results of continuous queries are often generated over a predefined window and then either stored, updated, or used to generate a new data stream of intermediate results.

Examples of DSMS include the following: Aurora [36], Telegraph[37], Gigascope [52], Nile [83], STREAM[11], Borealis [2], HiFi [70], IrisNet [73], Cougar [29], Apache S4 [130], T-Storm [190], C-MR [15], ESC [155], Apache Storm [1] and Tribeca [173]. The remainder of this section reviews the main data stream processing systems.

5.1.1 Classification of stream processing systems

As Fig.5.1 illustrates, stream processing systems can be classified based on three criteria [147]:

- The first criterion is concerned with the topology of a continuous query within a DSMS:
  - Workflow-based: A workflow-based system is based on a step-by-step execution model. Basically, in workflow-based systems, a query (set of operators) is turned into a workflow (set of nodes in specific order).
  - MapReduced-based: A MapReduce system uses a MapReduce paradigm to massively parallelize query processing.
  - Hybrid approach.
- The second criterion is concerned with a window incremental processing mechanism (time-based window) or there is support for window batch processing mechanisms (tuple-based window).
- The third criterion is concerned with whether the DSMS is centralized or distributed.

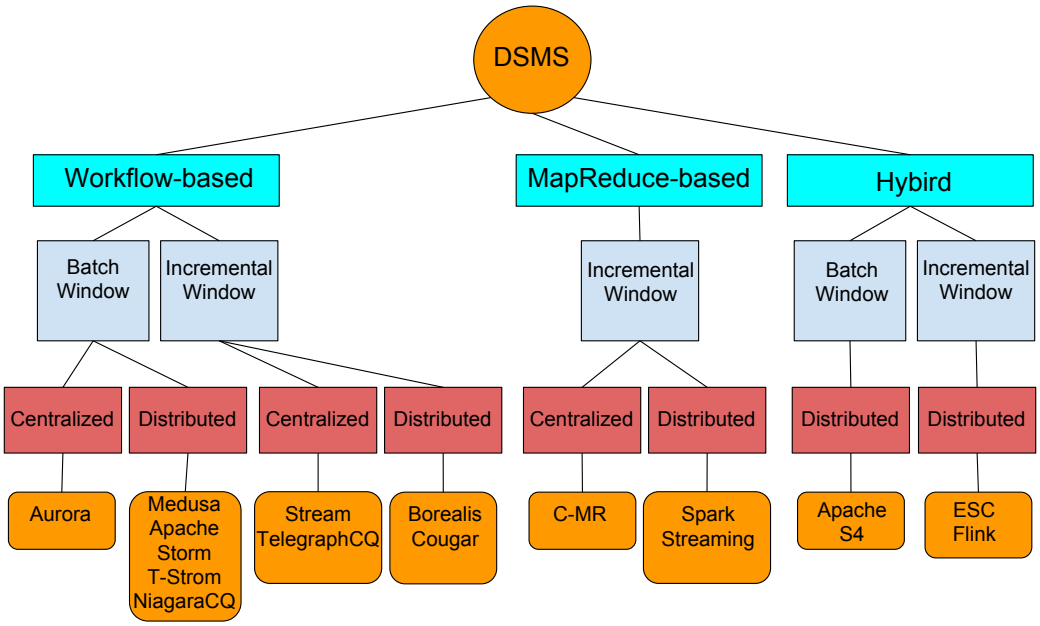

Figure 5.1: Data Stream Management Systems classification [147]

5.1.2 Representative Data Stream Processing Platforms

- **STREAM:** STREAM is Stanford's DSMS project that introduces a language specifically for querying stream. This is the continuous query language (CQL) which is an evolution of the SQL language by adding sliding window operators [11]. The idea is to represent streams by means of relations. Basically, the authors propose that the relevant

portion of the incoming stream that is queried is maintained like a relation. This allows the data to be queried by using traditional DBMSs queries. Finally, the resulting relational data is converted to a stream of tuples. STREAM constructs tree-shaped query plans. STREAM is focused on effectively processing data streams with bursty arrival rates. Basically, when the input rate is high, the system approximates query results after shedding some data. However, it does not support sharing across multiple queries.

- **Aurora:** Aurora [36], developed by Brown University and MIT, is a distributed stream processing network structure. Initially, Aurora was designed as a single site stream processing engine. Later, the Aurora functionality was combined with the Medusa [201] system to achieve distributed processing. In Aurora, a stream is modelled as a sequence of time-stamped tuples [5]. Aurora provides some processing in the neighbourhood of the data sources instead of transferring the generated raw data from data sources to a centralized processing environment such as a cloud. This approach can increase the efficiency of using of processing and network resources.

- **TelegraphCQ:** TelegraphCQ [37] is a continuous query processing system developed by the University of Berkeley. Operators, in TelegraphCQ, are called dataflow modules (TelegraphCQ nodes). TelegraphCQ nodes are assigned to different machines. A node, in TelegraphCQ, can receive several input data streams to be processed. The query engine of Telegraph is called Eddies.

- **Borealis:** Borealis [2] is a distributed stream processing system and is based on the data stream processing model of Aurora and Medusa. Borealis added the ability to modify queries during runtime [2].

- **IrisNet:** The IrisNet [73] provides an infrastructure that allows data consumers to access data from distributed sensors. IrisNet consists of a sensor agent nodes, and organizational agent nodes. Sensor agents are the nodes that provides an interface to collect data from sensors. Organizational agents are the nodes that provide distributed databases for storing observations from sensor agents.

- **Cougar:** Cougar [29] is a project from Cornell. Cougar is used for querying sensor networks. The objective of Cougar is to combine live data from a sensor network with required stored data by using using continuous queries.

- **NiagaraCQ:** NiagaraCQ [38] focuses on running continuous queries over XML data. The query language for XML data is called XML-QL.

- **Apache Storm:** Apache Storm [1] is a real-time stream processing framework built by Twitter. There are three main modules in an Apache Storm cluster: (i) Nimbus component; (ii) Supervisor component; (iii) ZooKeeper component.

- **Apache S4:** The Simple Scalable Streaming System (S4) [130] was developed by Yahoo. S4 is a distributed real-time stream processing system. S4 forms a network of processing elements. The processing elements are assigned to distributed processing nodes and processing nodes accept data streams.

5.2 Geo-Stream Management System (GSMS)

Continuing development in sensor technology makes it easier to gather geo-referenced data in real-time. Geo-referenced data streams are data streams with geographical (location) information [16][26]. Geo-referenced data streams are often generated by millions of users and devices [160][126][150][133] [92].

Many applications require the results of geo-streaming data to be timely. For example, a weather warning system watches special weather conditions and issues weather warnings (such as flood or tornado warnings). People can be notified of warnings via text messages [92]. For example, people within a certain distance of an evacuation area can be notified by text message.

5.2.1 Real-time Sensor Data Stream

A sensor data stream is a time series of sensor measurements which represents a tuple with a following schema $s_i =< t_s, l_{s_i}, v_1, v_2, .., v_n >$ where t_s and l_{s_i} represent the time-stamp and location of generated tuple by a sensor node s_i [133].

There may be a large number of sensor nodes within a geographic region which may result in sensor streams being spatially dense. The sensors can generate data at a high frequency. For example, the deployment of sensors in a city could vary between 200 to 20k sensors [133][126][150] and sample rates could vary between two seconds to every 10 minutes.

Due to the nature of the sensor networks, geo-sensor networks can be composed of streams from different devices which introduces varying accuracy that needs to be accounted for in analysis [129][167][74][99].

Real-time analysis of geo-referenced streams introduces new challenges. For example, analysis of geo-sensor data streams includes looking for patterns in the raw data, cross-correlating raw streams with other sensor streams, historic data and/or model predictions, and aggregating and summarizing raw sensor data. The following are requirements for designing a system for analyzing saptio- tempoeral streams of data in real-time [124].

- **Supporting huge volume of raw sensor streams analysis:** A data management system should be capable of providing efficient support for real-time queries over very large amounts of sensor data streams and be able to keep up with incoming data.

- **Integration with traditional data:** Since understanding, analyzing and interpreting current data is often performed through correlation with historic and/or model data, data management tools should provide seamless data representation and query capabilities between real-time sensor streams and historic and model data.

5.2.2 Sensor Data Stream Management Systems

In the following section, we introduce some of current data management approaches to support analysis of real-time sensor data streams.

Hadoop-GIS

Hadoop/GIS (geographic information systems) [57] tools are useful for spatial analysis of collected and stored data. Some tools such as [10][26] use Hadoop/GIS to integrate large stored data with the processing of real-time data [10][26].

Spatial and Spatio-Temporal Database Systems

Spatial-temporal database systems are widely used today [102]. These systems use data types to describe spatial data streams. A spatio-temporal database system uses database concepts for both space and time information [133]. Spatial-temporal systems are candidates for applications with sensor data streams that generate low volume of data [133] [113]. However, spatial-temporal database systems do not support continuous queries [69] and they are not designed for real time streaming applications.

5.3 Query Operator Placement

Distributed data stream processing systems (e.g., Borealis [2], Medusa [201], and IrisNet [73]), support applications with real-time processing requirements [4]. These data stream processing systems do this by pushing data streaming operators (query operators) away from a centralized processing point such as a cloud to try to reduce the communication network traffic. Pushing query operators into the network and away from a centralized processing node is known as query operator placement. A data streaming operator (query operator) is a function which is used to process an input data stream and produce output results [83][157]. Each operator has at least one input stream and one output stream. Typically these operators can take one or more input streams and produce one or more output streams. There is a body of work (e.g., [83][157][120][177][80]) that describes semantics for these operators. The basic data streaming operators are map, join, aggregate, filter and union.

The query operator placement problem is NP-hard. Several heuristics have been proposed. Several query operator placement approaches have been proposed in the literature (e.g., [175] [202] [191] [91] [93] [136] [144][168][60] [30]).

Huang et al. [93] model the relationship between the query operator execution time and an amount of remaining computing capacity on a resource node. They proposed a heuristic placement method to minimize the network usage. Pietzuch et al. [136] and Rizou et al. [144] proposed a decentralized placement algorithm to minimize network usage. Eidenbenz et al. [60] analysed the placement problem for a subset of distributed stream processing application.

The primary approach to minimize overall cost would be to place operators as close as possible to the data sources. However, data sources (e.g., sensors) are likely to have limited processing and storage capabilities. Previous work on operator placement problems for distributed query processing includes [175][35][144][136][93][167][117][63][118]. Some previous work (e.g., [117]) focuses on placement of aggregation operators, and does not consider filter operators or join operators.

5.4 Gap Analysis

This chapter presented the data stream processing systems and query operator distribution. The main problem that was addressed in this thesis was how to find the best point inside the network for placement of continuous query operators. In other words, we try to detect the distributed computational resources for hosting and executing the operator of the IoT application. The problems addressed in this thesis have been studied in the literature in the context of specific individual stream-based applications. For example, in the literature, distributed stream processing systems (e.g., [2][201][94][39][73]) show that moving query operators into the network (closer to the data sources) reduces the overhead in terms of latency and bandwidth consumption. Although, query operator placement is critical to improve the performance of data stream processing systems, the query operator placement is not addressed by current data stream processing systems [135] [136] [87][51][35].

Chapter 6

Query Operator Problem Formulation

This work looks to find the best point for placement of stream processing operator(s) to meet the requirements of real-time stream-based applications. A set of stream operators that are placed in a specific order to analyse the flow of data is known as a query graph. The flow of data can either come directly from a data source (e.g. sensor) or after one or more operators have been applied to the data from the data source. On the one hand, placing query operator(s) on computing nodes close to the network edge reduces transmission costs. On the other hand, placing query operators closer to computing resources at the network edge could result in high costs due to limited processing resources. The goal is to properly balance these opposing effects to minimize overall cost.

This chapter is organized as following: Section 6.1 reviews the impact of using fog platform on IoT application. Section 6.2 defines the query graph by using set notations and explains the query graph properties. Section 6.4 gives an analytical model for query graph operator placement by considering computational cost and communication cost. Section 6.3 discuses the motivation IoT application scenarios and query graph that are used for the motivation IoT application.

6.1 Reviewing the Impact of Using Fog Platform on IoT Applications

There is considerable work [72] [41] [174] [56] [125] [97] [127] [119] that describes the challenges in deploying IoT applications through the use of relying only on a cloud. These include satisfying QoS requirements which may be difficult to satisfy due to latency between IoT devices and the cloud and the high amount of bandwidth that is needed due to the large amount of data expected to be generated.

Researchers have studied the effectiveness of using the fog node for specific use cases. Brogi et al [32] proposed that fog nodes can reduce the amount of data sent to the cloud through filtering the generated data streams and that analytic operations be processed on the fog node. Hassanalieragh et al [84] proposed the use of a cloudlet for applying an aggregation operation to sensor data. The result is sent to the cloud. Yue et al [197] associates a cloudlet with a community where a community consists of users with common interests in specific data e.g., transportation data. Satyanarayanan et al [152] introduced a multilevel cloudlet hierarchy

expansion to the basic cloudlet option. There are no experiments that evaluate the effectiveness.

Fesehaye et al [66] studied the impact of cloudlets for mobile applications that included file editing, video streaming and collaborative chatting. Their experimental results show that cloudlets outperform clouds provided that the number of hops to the data sources is small. This work does not consider IoT applications. Experimental work by Mahmud et al [119] suggests that the effectiveness of using fog nodes is limited by the number of sensors i.e. fog nodes are effective until the number of sensors produces so much data that it cannot be accommodated by a fog node and thus must be sent to the cloud. Maier et al [121] shows that a two-level cloudlet and cloud architecture can provide good response times for applications for heavy traffic loads. Most of the existing work motives the use of having computing nodes close to the data sources but there is relatively little work on evaluating the effectiveness of different organization of multiple fog nodes.

The query operator placement problem has been widely studied in the literature under different modeling assumptions and optimization goals (e.g., [175] [202][191][91][93][136][144] [168][60]). Since the query operator placement problem is NP-hard, several heuristics have been proposed [136][51][35].

6.2 Definition of a Query Graph

A query operator is a function which is used to process an input data stream and produce a result [83][157]. Each query operator has at least one input stream and one output stream. Data stream operators are categorized as stateless or stateful. Stateless operators process each input tuple individually and the corresponding output is generated without maintaining any state. Stateful operators maintain state as they process multiple input tuples in order to produce the result. For example, aggregate operators are stateful. A filter operator is an example of a stateless operator. A query for processing a stream of data is represented by a query graph and a query graph consists of a set of query operators. In the other words, a query receives an input, applies an operation to produce an output. A query graph is represented by $G_Q = (V_Q, E_Q)$ where V_Q represents a set of queries: $q_0, q_1, q_2, \ldots, q_{n-1}$. An edge $(q_i, q_j) \in E_Q$ means that q_i's output is an input to q_j.

6.3 Example Application Scenarios and Query Graphs

This section presents four application scenarios. Two of the applications require data processing and decision making based on data from local data sources. The other two applications require geospatial analysis. Geospatial analysis is a process of gathering and analysis of the data with geographical (location) information [16][26]. In this section we present several applications an their query graphs. These will be used to illustrate query graphs and in the experiments presented in chapters 7 and 9.

6.3.1 Local Congested Highway Notification Scenario

A fog node is assigned to a highway segment. Each fog node periodically receives reports from each vehicle currently in the segment assigned to the fog node. The fog node uses these reports to determine if there is congestion on that segment. If there is congestion then the fog node sends a notification to the vehicles in its segment as well as to the other fog nodes. Figure 6.1 illustrates a query graph that is used for detecting traffic congestion.

- Query Q_1 receives a stream of positions reports from vehicles. Query Q_1 applies an operation that returns the segment number from longitude and latitude information from a position report.
- Query Q_2 returns the number of vehicles in a segment within a window of time.
- Query Q_3 returns the summation of vehicles' speed in a segment within a window of time.
- Query Q_4 returns the average of vehicles' speed in one segment within a window of time.
- Query Q_5 takes the average speed of one segment form query Q_4 and generates the congestion notification message if the average speed of that segment is less than a threshold value.

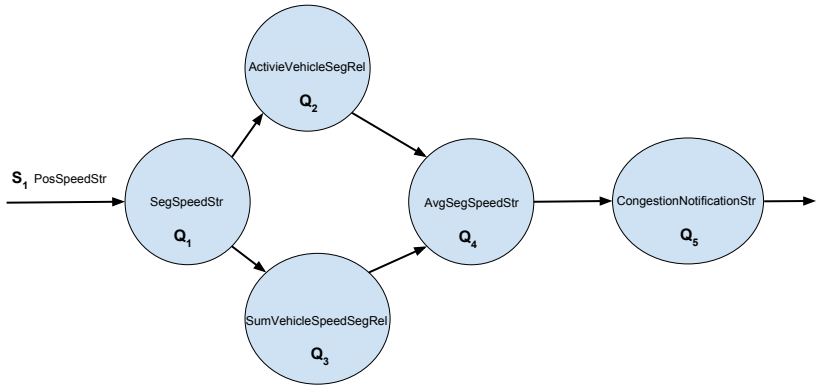

Figure 6.1: Query Graph for Local Traffic Congestion Notification Scenario

6.3.2 Local Camera Surveillance Scenario

In this scenario the goal is to measure the average crowd size by using cameras. One of the goals of crowd size measurement is to ensure safety and security in public in the case of emergency evacuations (in the event of fire, natural disaster) [143]. For example, in the event of an emergency such as a fire we require an estimate of crowd size to provide a reasonable estimate of people for emergency evacuations and evacuation path planning. Camera video streams can be used to determine the crowd size. Periodically a camera sends a tuple of data to a fog node or a cloud. Each tuple consist of a timestamp, camera location information as identified by

longitude and latitude, camera identifier, and a sequence of video frames taken over a specific period of time. A fog node is assigned to a street intersection. A fog node accepts connections from more than one camera but each camera only connects to one fog node. Each intersection can have one or more cameras. Figure 6.2 illustrates a query graph that is used for measurement of crowd size.

- Query Q_1 receives stream of video from camera(s) for an intersection. Query Q_1 applies an operation that returns I frames corresponding to an intersection.
- Query Q_2 returns the list of camera(s) for an intersection over a window of time.
- Query Q_3 measures the number of people for an intersection over a window of time by analysing the stream of I frames for an intersection over a window of time. Query Q_3 receives the stream of I frames from query Q_1.
- Query Q_4 calculates the average crowd size of people.

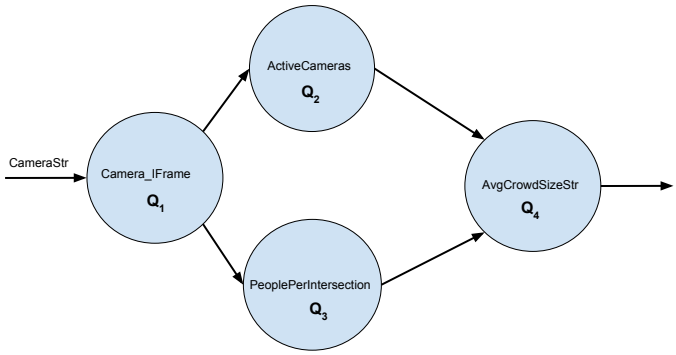

Figure 6.2: Local Camera Crowd Size Scenario Query Graph

6.3.3 Regional Congestion Traffic Notification

Each vehicle has a sensor that periodically emits a position report that identifies the vehicle's location [12]. The position reports can be used to determine if there is congestion. If there is congestion then vehicles can be notified so that the driver can reroute. Re-routing requires information from multiple areas since traffic congestion can occur over multiple areas. The approach we take is based on calculating traffic congestion for a segment of a highway. Traffic is calculated for larger areas based on segment traffic. Segments can be determined based on the longitude and latitude found in the position report. Traffic congestion for a segment is based on the calculation of the average speed over a window of time. The average speed of a region consists of the average speed of vehicles of all segments in that region. When congestion is detected in a region then a congestion notification message is sent to vehicles to re-route. Fog nodes provide an alternative route to the vehicles. Figure 6.3 illustrates a query graph that is used for detecting traffic congestion.

- Query Q_1 receives a stream of positions reports from vehicles. Query Q_1 applies an operation that returns the segment number from longitude and latitude information from the position report.
- Query Q_2 returns the number of vehicles in a segment within a window of time.
- Query Q_3 returns the summation of vehicles' speed in a segment within a window of time.
- Query Q_4 returns the average of vehicles' speed in one segment within a window of time.
- Query Q_5 returns the number of active segments of highways that an aggregation needs to be done for.
- Query Q_6 takes the average speed of multiple segment form query Q_4 and generates the congestion notification message if the average speed of the multiple segments is less than a threshold value. Query Q_6 obtains the list of multiple segments from query Q_5.
- Query Q_7 generates congestion notification if the average speed for the multiple segments is less that a certain speed.

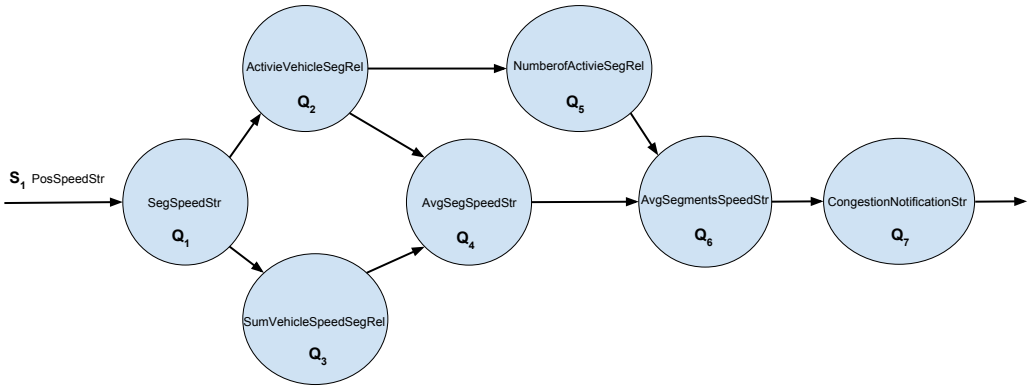

Figure 6.3: Query Graph for Traffic Congestion Notification Scenario

6.3.4 Regional Camera Crowd Size Measurement Scenario Results

In this scenario the goal is to measure the average crowd size of people by using cameras in a region. Video streams from multiple cameras can be used to determine the crowd size of people in a region. Cameras send a tuple of data to a fog node or a cloud periodically. Each tuple consist of a timestamp, camera location information as identified by longitude and latitude, camera identifier, and a sequence of video frames taken over a specific period of time. An intersection represents the minimal unit for which crowd size of people is calculated. Intersections can be determined based on the longitude and latitude found in the tuple sent by the cameras. Each intersection is associated with a camera. A region includes multiple intersections. A region has boundaries that can be represented in terms of longitudes and latitudes [88]. Figure 6.4 illustrates a query graph that is used for measurement of crowd size.

- Query Q_1 receives stream of video from camera(s) for an intersection. Query Q_1 applies an operation that returns I frames corresponding to an intersection.
- Query Q_2 returns the list of camera(s) for each intersection over a window of time.
- Query Q_3 measures the number of people for an intersection over a window of time by analysing the stream of I frames for an intersection over a window of time. Query Q_4 receives the stream of I frames from query Q_1.
- Query Q_4 calculates the average crowd size.
- Query Q_5 returns the list of intersections in a region.
- Query Q_6 calculates the average number of people for the intersections in a region over a window of time.

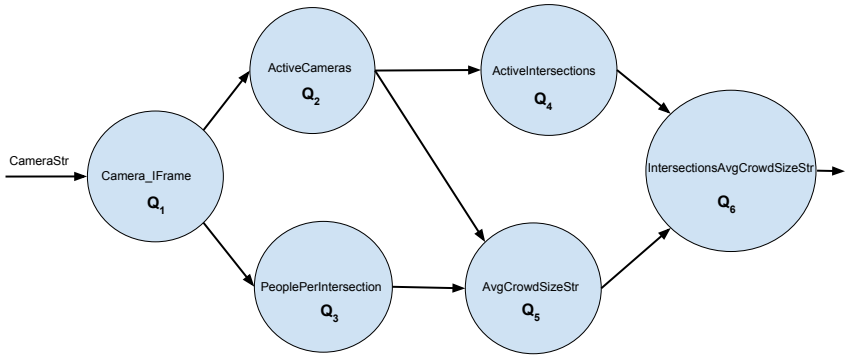

Figure 6.4: Camera Crowd Size Scenario Query Graph

6.4 Query Plan Embedding

A guest graph is a query graph that represented by $G_{guest} = (V_{guest}, E_{guest})$ and a host graph is a distributed processing environment consisting of fog nodes that is represented by $G_{host} = (V_{host}, E_{host})$ where $E_{host} = \{(u, v, d)\}$ where d is the underlay path distance in terms of underlay hops between u and v for $u, v \in V_{host}$ and d is calculated by using Equ.4.8. The embedding of a given guest graph into a host graph is defined by an injective function $M : G_{guest} \rightarrow G_{host}$ [27]. The mapping function M assigns to each edge $(u, v) \in E_{guest}$ a path in G_{host}. Embedding a given guest graph into a host graph can be formulated as an optimization problem that has been shown to be NP-complete [180][71][65].

6.4.1 Embedding Distortion Computation

The distortion (or dilation) of an embedding is defined as the difference in the lengths of the shortest path between two vertices in the guest graph and the shortest path between these vertices after embedding into the host graph. Let $D(u, v)$ be the distance function, which is the

shortest path distance between u and v for every pair of vertices $u, v \in G_{guest}$. The distortion of the embedding indicate the communication slowdown caused by embedding. The distortion of an embedding M is calculated as follows [68]:

$$d_M = max_{u,v \in V_{guest}}(\frac{D(u,v)}{D(M(u), M(v))}) \tag{6.1}$$

where $M(u), M(v) \in G_{host}$. The value of d_M is between 0 and 1. For a mapping function M if d_M is closer to 1 then we can conclude that M maps the query operator close to the data source in terms of underlay hops. For a mapping function M if d_M is closer to 0 then we can conclude that M maps the query operator close to the network core in terms of hops.

6.4.2 Cost of Embedding a Query Graph

The cost calculation for embedding a host query graph is based on the following: (i) the relative amount of available bandwidth (network traffic) and (ii) the available computational power on fog nodes. A weighting factor is required to determine which part of the cost function should be more dominant. The cost of embedding a query graph by using an embedding function M into the fog nodes is calculated by using Equ 6.2.

$$\min \sum_{\substack{u,v \in V_{guest} \\ e=(u,v)}} C(e) \tag{6.2}$$

where $C(e)$ is the cost function and $e \in E_{guest}$. $C(e)$ is defined by using Equ 6.3[167].

$$C(e) = \sum_{u,v \in V_{guest}, x \in V_{host}} \Big(\psi \times T_x + (1-\psi) \times \sum_{e' \in D(M(u),M(v))} \frac{D_{tx}}{\omega(e')}\Big) \tag{6.3}$$

where $D(M(u), M(v))$ represents the shortest path between $M(u)$ and $M(v)$ and ω represents the minimum available amount of bandwidth on the shortest path between $M(u)$ and $M(v)$. A weight factor ψ has been introduced to determine which part of the cost function should be more dominant. D_{tx} is the sum of all data transmitted by each operator running at fog node $x \in V_{host}$ and is calculated by using Equ 6.4. T_x represents the amount of time that is required to process received data at the fog node x and is calculated by using Equ 6.5. Then we select an embedding, among all possible embeddings, that introduces the smallest distortion factor (by Equ. 6.1). The calculation of D_{tx} and T_x is explained [4][107][167][68].

For each fog node $x \in V_{host}$, the data transmitted D_{tx} is the sum of all data transmitted by each operator hosted on fog node x. The amount of data transmitted from a fog node x is calculated by Equ. 6.4. Each operator o_i (or each query vertex) transmits its output to all of its children.

$$D_{tx} = \sum_{\{o_i : M(o_i)=x\}} output(o_i) \times |S_i| \tag{6.4}$$

where $|S_i|$ is the cardinality of the successors for operator o_i and $output(o_i)$ be the amount of data generated by o_i.

We assume that E_x is the execution power of a fog node $x \in V_{host}$. Therefore, the amount of time that is required to process received data at the fog node x is calculated by using Equ. 6.5.

$$T_x = \frac{E_{rx}}{E_x} \tag{6.5}$$

where E_{rx} represents the amount of required processing power which is required to process received data at a fog node $x \in V_{host}$. E_{rx} is calculated by using Equ 6.6.

$$E_{rx} = \sum_{o_i \in P_i} E(o_i) \tag{6.6}$$

where $E(o_i)$ is a function that measures the required processing power for o_i.

Impact of Weight Factor ψ in the Cost Function

For embedding a query graph to distributed fog nodes we make the assumption that the distributed fog nodes that are closer (in terms of physical network hop distance) to the edge of the network (data sources) have less computational power compared to those closer to the network core. Also, fog nodes closer to the core of the network have more processing power compared to fog nodes close to the network edge. For embedding a query graph the cost function which is represented by Equation 6.3 is considered. We can consider the following embedding based on the weight factor ψ:

- $\psi = 0$**:** In this case the entire query graph is embedded to fog nodes at the edge of the network. In the other words, the cost function considers the communication cost only. Considering the query graph for local congestion highway notification scenario. The query graph includes five query vertices Q_1, Q_2, Q_3, Q_4, and Q_5. In this case all of Q_1, Q_2, Q_3, Q_4, and Q_5 are mapped to fog nodes in the neighborhood of data sources to minimize cost.

- $\psi = 1$**:** In this case the entire query graph is embedded to a cloud. In the other words, the cost function considers only processing cost. Considering the query graph for the local congestion highway notification scenario. The query graph includes five query vertices Q_1, Q_2, Q_3, Q_4, and Q_5. In this case all of Q_1, Q_2, Q_3, Q_4, and Q_5 are mapped to a cloud to minimize the processing cost.

- $0 < \psi < 1$**:** In this case, as the value of ψ approaches 1, the query vertices are mapped to the fog nodes closer to the network core and as the value of ψ approaches 0 then the query vertices are mapped to the fog nodes close to the network edge. In other words, the cost function considers the sum of processing cost and communication cost together. For the local congestion highway notification scenario the query graph includes five query vertices Q_1, Q_2, Q_3, Q_4, and Q_5. In this case Q_1, Q_2, Q_3, Q_4, and Q_5 are mapped to fog nodes in such a way to minimize the communication cost.

Chapter 7

Study the Impact of Using Fog Platform on IoT Applications

This chapter presents the results of experimental studies that compares the use of using one or more levels of fog nodes for high and low volumes of data. We used the applications described in Section 6.3.

This chapter is organized as follows: Section 7.1 presents the performance metrics that are used for evaluation. The experimental environment is described in Section 7.2. Section7.3 presents the results for the local congested highway notification scenario. Section 7.4 presents the results for local crowd size measurement using camera surveillance. Section 7.5 presents the results for the regional congestion traffic notification scenario. Section 7.6 presents the results for the regional camera surveillance scenario.

7.1 Performance Metrics

This section describes the evaluation metrics.
Execution Time. The execution time consists of two components as follows:

- **Buffering Time:** The buffering time is the amount of time that a tuple is kept in the buffer of the fog node before it is processed.
- **Time to Perform a Task:** This is the amount of time that is spent by a processing unit to perform an analysis on a received stream of data.

End-to-End Delay: The end-to-end delay refers refers to the time taken for a packet to be transmitted across a network from source to destination [3]. End-to-end delay is also referred to as network latency.
Response Time: The response time is defined as the time between the generation of a tuple and the delivery of results of the analysis of the tuple. The response time consists of execution time and end-to-end delay.
Distortion: Distortion (or tuple miss ratio) is a performance metric that represents the percentage of tuples that are lost compared to the total number of generated tuples. The distortion rate consists of two parts: (1) Tuple loss due to the loss in the communication network (UDP

protocol was the transportation protocol); and (2) Loss from time-out. Time-out is defined as a specified period of time that will be allowed to elapse for a tuple to reach the sender.

7.2 Experimental Environment

The experimental environment is a set of networked fog nodes that are organized hierarchically. Fog nodes closer to the data sources have less computational power compared to those closer to the network core [194][128]. The number of hops between data sources and a cloud is between 28 to 32 and the number of hops between data sources and fog nodes is between 4 to 6. In this simulations for level-0 fog nodes the CPU power is 1.7K MIPS, the CPU power for fog nodes in level 1 is 2.7K MIPS, and the CPU power of the cloud is 48K MIPS [179].

7.3 Results of Local Congested Highway Notification

This section presents the evaluation results for the Local Congested Highway Notification scenario. A position report tuple includes the following attributes: timestamp, longitude info, latitude info, direction, vehicle identifier, speed, and highway number. The size of the position report tuple that is able to include these attributes is 210 bytes with average size of an attribute being 30 bytes [96]. For local congested highway notification all the operators in queries Q_1, Q_2, Q_3, Q_4, and Q_5 are being placed in level 0 fog nodes and when using the cloud all the operators in queries Q_1, Q_2, Q_3, Q_4, and Q_5 are placed in the cloud.

Table 7.1 shows that increasing the number of fog nodes decreases the execution time as the number of fog nodes and vehicles increases. From the results in Table 7.1, we conclude that a sufficient number of fog nodes close to the data sources results in better execution time than the use of a cloud.

Table 7.1: Average Execution Time

#Vehicles / Number of Fog Nodes	300 Vehicles	600 Vehicles	1500 Vehicles	3000 Vehicles
3	48.753 (s)	311.152 (s)	15731.719(s)	43481.013 (s)
6	22.981 (s)	40.86 (s)	2889.138 (s)	16731.023 (s)
9	15.509 (s)	32.421 (s)	76.911 (s)	7966.051 (s)
12	9.891 (s)	23.031 (s)	55.428 (s)	3065.892 (s)
30	4.330 (s)	9.159 (s)	20.981 (s)	44.043 (s)
60	1.882 (s)	4.102 (s)	10.827 (s)	21.221 (s)
Cloud	3.509 (s)	5.908 (s)	12.199 (s)	24.581 (s)

Table 7.2 shows that as the number of fog nodes increases the end-to-end delay decreases since fewer tuples are being sent to each fog node.

Table 7.3 shows that a sufficient number of fog nodes close to the data sources results in better response time than the use of a cloud. Table 7.4 shows that as the number of fog nodes increases the distortion decreases. Table 7.5 shows the average distortion due to network latency as the number of fog nodes and vehicles varies. The results reflect the small amount of data in a position report.

The results shows that increasing the number of fog nodes results in shorter execution times and less distortion for larger amounts of data.

Table 7.2: Average End-to-End Delay

#Vehicles / Number of Fog Nodes	300 Vehicles	600 Vehicles	1500 Vehicles	3000 Vehicles
3	0.00214 (s)	0.00412 (s)	0.01723 (s)	0.10612 (s)
6	0.00164 (s)	0.00217 (s)	0.00536 (s)	0.01173 (s)
9	0.00054 (s)	0.00139 (s)	0.00373 (s)	0.00785 (s)
12	0.00053 (s)	0.00123 (s)	0.00264 (s)	0.00533 (s)
30	0.00021 (s)	0.00044 (s)	0.00103 (s)	0.00215 (s)
60	0.00013 (s)	0.00021 (s)	0.00055 (s)	0.00109 (s)
Cloud	0.03743 (s)	0.08424 (s)	0.19727 (s)	0.39432 (s)

Table 7.3: Average Response Time

#Vehicles / Number of Fog Nodes	300 Vehicles	600 Vehicles	1500 Vehicles	3000 Vehicles
3	48.75514 (s)	311.15612 (s)	15731.73623 (s)	43481.11912 (s)
6	22.98264 (s)	40.86217 (s)	2889.14336 (s)	16731.03473 (s)
9	15.50954 (s)	32.42239 (s)	76.91473 (s)	7966.05885 (s)
12	9.89153 (s)	23.03223 (s)	55.43064 (s)	3065.89733 (s)
30	4.33021 (s)	9.15944 (s)	20.98203 (s)	44.04515 (s)
60	1.88213 (s)	4.10221 (s)	10.82755 (s)	21.22209 (s)
Cloud	3.54643 (s)	5.99224 (s)	12.39627 (s)	24.97532 (s)

Table 7.4: Distortion Due to Buffering

#Vehicles / Number of Fog Nodes	300 Vehicles	600 Vehicles	1500 Vehicles	3000 Vehicles
3	0 %	18.01 %	49.78 %	51.08 %
6	0 %	0 %	45.73 %	47.01 %
9	0 %	0 %	0 %	44.17 %
12	0 %	0 %	0 %	40.18 %
30	0 %	0 %	0 %	0 %
60	0 %	0 %	0 %	0 %
Cloud	0 %	0 %	0 %	0 %

Table 7.5: Distortion Due to Network Latency

#Vehicles / Number of Fog Nodes	300 Vehicles	600 Vehicles	1500 Vehicles	3000 Vehicles
3	0 %	0 %	0 %	0 %
6	0 %	0 %	0 %	0 %
9	0 %	0 %	0 %	0 %
12	0 %	0 %	0 %	0 %
30	0 %	0 %	0 %	0 %
60	0 %	0 %	0 %	0 %
Cloud	0 %	0 %	0 %	0 %

7.4 Results of Local Crowd Size Measurement with Camera Surveillance

In this section, we show the results for the Local Camera Surveillance Scenario. The number of cameras, in total, varies as follows: 9, 18, 27, 36, 135, and 180. It was assumed that there are three areas under camera surveillance. An area is defined using four coordinates where each coordinate is associated with a longitude and latitude values. An area includes intersections and there is a camera associated with an intersection. A fog node receives the video stream

and calculates the crowd size by analysing the video frames. The number of fog nodes varies as follows: 3, 6, 9, 12, 30, 60. The size of a video tuple varies between 190 KB and 204 KB. Each tuple consists of a timestamp, camera longitude information, camera latitude information, camera identifier, frame number, I frame, B frame, P frame. The experimental environment is the same as it was explained in Section 7.2. The amount of required processing power to process each video chunk is 4500 MIPS [79][140]. Each chunk includes 24 frames. For local crowd size measurement with camera surveillance all the operators in queries Q_1, Q_2, Q_3, and Q_4 are placed in a level 0 fog nodes when only fog nodes are being used and when only the cloud is used all the operators in queries Q_1, Q_2, Q_3, and Q_4 are placed in the cloud.

The results in Table 7.6 shows that increasing the number of fog nodes results in better execution time. As the number of fog nodes increases for a larger number of cameras, the execution time is significantly better than using only the cloud.

Table 7.6: Average Execution Time

#Cameras in Total / Number of Fog Nodes	9 Cameras	18 Cameras	27 Cameras	36 Cameras	135 Cameras	180 Cameras
3	89.081 (s)	1915.554 (s)	3775.503 (s)	7434.134 (s)	25977.594(s)	33480.366(s)
6	59.522 (s)	89.095 (s)	1297.455 (s)	1915.622 (s)	12448.445 (s)	16785.665(s)
9	30.393 (s)	58.996 (s)	88.443 (s)	678.099 (s)	7492.288(s)	10590.233(s)
12	28.124 (s)	58.865 (s)	57.933 (s)	88.385 (s)	5634.838 (s)	7492.342(s)
30	28.083 (s)	28.969 (s)	29.023 (s)	58.328 (s)	1297.633 (s)	1915.421 (s)
60	27.892 (s)	28.043 (s)	28.299 (s)	28.523 (s)	88.955(s)	89.554 (s)
Cloud	9.453 (s)	17.933 (s)	28.303 (s)	37.766 (s)	1176.774(s)	2163.828(s)

Table 7.7 shows that as the number of fog nodes increases the end-to-end delay decreases when the number of cameras increases.

Table 7.7: Average End-to-End Delay

#Cameras in Total / Number of Fog Nodes	9 Cameras	18 Cameras	27 Cameras	36 Cameras	135 Cameras	180 Cameras
3	0.0431(s)	0.0912 (s)	0.1273 (s)	0.1783 (s)	0.6291(s)	0.8376 (s)
6	0.0131(s)	0.0423 (s)	0.0705 (s)	0.0852 (s)	0.3367(s)	0.4278(s)
9	0.0130(s)	0.0285 (s)	0.0412 (s)	0.0541 (s)	0.2096(s)	0.2785(s)
12	0.0128(s)	0.0284 (s)	0.0285 (s)	0.0426 (s)	0.1593(s)	0.1992(s)
30	0.0127(s)	0.0129 (s)	0.0129 (s)	0.0285(s)	0.0705 (s)	0.0843(s)
60	0.0127(s)	0.0128(s)	0.0129 (s)	0.0127 (s)	0.0418 (s)	0.0416(s)
Cloud	0.4635 (s)	0.8272 (s)	1.3865 (s)	1.8418 (s)	6.9233(s)	9.2412(s)

Table 7.8 shows that increasing the number of fog nodes decreases the average response time as the number of fog nodes and vehicles increase and the average response time is significantly less than the use of the cloud.

Table 7.9 shows that as the number of fog nodes increases the distortion decreases. Table 7.9 also shows that increasing the number of fog nodes results in a smaller distortion than relying only on a cloud for larger number of cameras.
Table 7.10 shows a significant difference for distortion due to network latency between using only the cloud and a set of fog nodes for a large number of cameras.

Overall the results show that relying only on the use of a cloud can be bottleneck for a large number of cameras. The results show that increasing the number of fog nodes results in shorter execution times. Although most of the reduction is due to decreasing the response time by

Table 7.8: Average Response Time

#Cameras in Total / Number of Fog Nodes	9 Cameras	18 Cameras	27 Cameras	36 Cameras	135 Cameras	180 Cameras
3	89.1241 (s)	1915.6452 (s)	3775.6303 (s)	7434.3123 (s)	25978.2231(s)	33481.2036(s)
6	59.5351 (s)	89.1373 (s)	1297.5255 (s)	1915.7072 (s)	12448.7817 (s)	16786.0928(s)
9	30.406 (s)	59.0245 (s)	88.4842 (s)	678.1531 (s)	7492.4976(s)	10590.5115(s)
12	28.1368 (s)	58.8934 (s)	57.9615 (s)	88.4276 (s)	5634.9973 (s)	7492.5412(s)
30	28.0957 (s)	28.9819 (s)	29.0359 (s)	58.3565 (s)	1297.7035 (s)	1915.5053 (s)
60	27.9047 (s)	28.0558 (s)	28.3119 (s)	28.5357 (s)	88.9968(s)	89.5956 (s)
Cloud	9.9165 (s)	18.7602 (s)	29.6895 (s)	39.6078 (s)	1183.6973(s)	2173.0692(s)

Table 7.9: Average Distortion Due to Buffering

#Cameras in Total / Number of Fog Nodes	9 Cameras	18 Cameras	27 Cameras	36 Cameras	135 Cameras	180 Cameras
3	44.94 %	49.76 %	49.88 %	49.93 %	49.98 %	50.19 %
6	34.74 %	44.24 %	49.69 %	49.76 %	49.31 %	49.97 %
9	33.72 %	42.37 %	44.94 %	47.33 %	48.92%	48.95%
12	33.15 %	42.21 %	42.37 %	44.94 %	48.53 %	48.84%
30	30.88 %	33.14 %	34.72 %	42.37%	47.65 %	47.96%
60	29.94 %	29.92 %	29.84 %	29.93 %	44.19%	44.89%
Cloud	2.35 %	26.11 %	34.6 %	37.22 %	49.12 %	49.32%

Table 7.10: Average Distortion Due to Network Latency

#Cameras in Total / Number of Fog Nodes	9 Cameras	18 Cameras	27 Cameras	36 Cameras	135 Cameras	180 Cameras
3	24.66 %	37.5 %	41.65 %	43.56 %	48.33 %	48.75 %
6	0 %	25.02 %	35.17 %	37.56 %	46.87 %	47.5 %
9	0 %	13.11 %	25.77 %	30.55 %	45.66%	46.11%
12	0 %	12.88 %	13.74 %	25.76 %	43.35%	45.33%
30	0%	0 %	0 %	13.01%	35.71%	37.55 %
60	0%	0 %	0 %	0%	25.68%	25.75%
Cloud	47.23 %	48.8 %	49.26 %	49.34%	49.84%	49.85 %

increasing the number of fog nodes, we also see that for a high number of cameras the network latency is also much smaller due to much less data being transferred to the cloud.

7.5 Results of Regional Congestion Traffic Notification

This section presents the evaluation results for Regional Congested Highway Notification. This section first describes the experimental configuration while later this Section presents the results.

7.5.1 Experimental Configurations

The following configurations of fog nodes were used in the experiments.

Configuration 1 assumes only the use of the cloud. Configuration 2 uses a Cloud and level 0 fog nodes. In configuration 2 each level 0 fog node measures the average speed for vehicles currently in the segment assigned to the fog node with results sent to the cloud. The cloud measures the average speed and identifies traffic congestion in a region by using information received from fog nodes in level 0. The cloud determines if the segment is inside the region by

comparing the longitudes and latitudes of segment and the region boundaries. Configurations 3, 4, 5, and 6 use two levels of fog nodes. Level-0 fog nodes directly receive data from data sources while level-1 fog nodes receive data from level-0 fog nodes. Each level-1 fog node measures the average speed for vehicles currently in the segment assigned to the fog node and sends results to a fog node in level 1. Each level 1 fog node measures the average speed for a larger region by using information received from fog nodes in level 0 in order to identify multiple segment traffic congestion. The difference between configurations 3, 4, 5, and 6 are the number of level-1 fog nodes. Configuration 3 has one level-1 fog nodes, configuration 4 has two level-1 fog nodes, configuration 5 has 3 level 1 fog nodes, and configuration 6 has 10 level-1 fog nodes.

The number of fog nodes in level 0 for each segment of a highway varies as follows: 2, 4, 10, 20, 50, 60, 80. There is one cloud. The number of vehicles for each level-0 fog node varies as follows: 25, 50, 75, 100, 125, 150, 200, 250, 375 , 500, 750, and 1000. Therefore, the total number of vehicles for each segment varies as follows: 500, 1000, 1500, and 2000.

For configuration 1 all the operators in queries Q_1, Q_2, Q_3, Q_4, Q_5, Q_6, and Q_7 are placed in a cloud. For configuration 2 the query operators in queries Q_1, Q_2, Q_3, and Q_4 are mapped to level 0 fog nodes and the query operators in queries Q_5, Q_6, Q_7 are mapped to a cloud. For configurations 3, 4, 5, and 6 the query operators in queries Q_1, Q_2, Q_3, and Q_4 are mapped to level 0 fog nodes and the query operators in queries Q_5, Q_6, Q_7 are mapped to level 1 fog nodes.

Table 7.11: Maximum Mapping Distortion for Congested Highway Notification Scenario

Config #	mapping distortion
Config 1	0.08
Config 2	0.08
Config 3,4,5,6	0.66

7.5.2 Results of Configuration 1, 2, 3, 4, 5, and 6

Table 7.12 shows the results for execution time. Configuration 1 consistently outperforms all configurations (2,3,4,5, and 6) when the number of level 0 fog nodes is 20 or less. However, configuration 6 outperforms the use of only the cloud for a higher number of level-0 fog nodes.

The results in Table 7.13 show that the use of only the cloud (configuration 1) has a higher end-to-end delay than configurations 2,3,4,5, and 6 but this delay represents a small part of the response time results presented in Table 7.14 which shows that response time has similar characteristics as execution time.

We conclude that more fog nodes that concurrently analyze data close to the data sources leads to better response time than relying on the cloud. This is the result of introducing a large amount of computing power close to the data sources.

Table 7.15 shows the average distortion due to buffering for configurations 1, 2, 3, 4, 5, and 6. Table 7.15 shows that as the number of fog nodes increases the distortion decreases which improves the quality of service. A data stream tuple is placed in a fog node's buffer upon its arrival. The fog node reads from the buffer and processes the tuples. When the number of

Table 7.12: Average Execution Time-Configuration 1, 2, 3, 4, 5, and 6

Number of Level 0 Fog Nodes	Config #	500 vehicles	1000 vehicles	1500 vehicles	2000 vehicles
2 fog nodes	Config 1	8.54 (s)	15.72 (s)	24.63 (s)	32.83 (s)
	Config 2	3068.81 (s)	16938.67 (s)	30806.55 (s)	44673.01 (s)
	Config 3	3069.33 (s)	16939.21 (s)	30808.64 (s)	44676.64 (s)
	Config 4	3069.185 (s)	16938.285 (s)	30808.018 (s)	44674.178 (s)
	Config 5	3068.681 (s)	16936.085 (s)	30806.508 (s)	44674.018 (s)
	Config 6	3068.01 (s)	16936.35 (s)	30806.77 (s)	44673.78 (s)
4 fog nodes	Config 1	8.54 (s)	15.72 (s)	24.63 (s)	32.83 (s)
	Config 2	58.53 (s)	3069.34 (s)	10003.44 (s)	16935.32 (s)
	Config 3	60.42 (s)	3070.81 (s)	10004.25 (s)	16940.22 (s)
	Config 4	60.075 (s)	3070.075 (s)	10003.573 (s)	16939.037 (s)
	Config 5	59.085 (s)	3068.883 (s)	10001.386 (s)	16937.917 (s)
	Config 6	58.21 (s)	3067.02 (s)	10001.26(s)	16937.35 (s)
10 fog nodes	Config 1	8.54 (s)	15.72 (s)	24.63 (s)	32.83 (s)
	Config 2	22.59 (s)	45.5 (s)	58.11 (s)	292.54 (s)
	Config 3	25.41 (s)	48.16 (s)	60.51 (s)	295.91 (s)
	Config 4	23.752 (s)	46.252 (s)	58.902 (s)	294.912 (s)
	Config 5	22.451 (s)	45.045 (s)	55.465 (s)	292.716 (s)
	Config 6	20.61 (s)	42.56 (s)	54.62 (s)	287.69 (s)
20 fog nodes	Config 1	8.54 (s)	15.72 (s)	24.63 (s)	32.83 (s)
	Config 2	11.35 (s)	22.33 (s)	34.23 (s)	45.85 (s)
	Config 3	17.91 (s)	28.41 (s)	40.54 (s)	52.07 (s)
	Config 4	14.895 (s)	25.235 (s)	37.315 (s)	48.335 (s)
	Config 5	13.421 (s)	24.472 (s)	36.432 (s)	47.492 (s)
	Config 6	10.11 (s)	19.43 (s)	33.49 (s)	42.02 (s)
50 fog nodes	Config 1	8.54 (s)	15.72 (s)	24.63 (s)	32.83 (s)
	Config 2	5.58 (s)	10.67 (s)	14.58 (s)	19.53 (s)
	Config 3	16.07 (s)	23.34 (s)	27.24 (s)	32.41 (s)
	Config 4	11.585 (s)	16.117 (s)	20.755 (s)	25.384 (s)
	Config 5	9.255 (s)	13.864 (s)	18.425 (s)	23.601 (s)
	Config 6	4.94 (s)	12.03 (s)	16.38 (s)	24.98 (s)
60 fog nodes	Config 1	8.54 (s)	15.72 (s)	24.63 (s)	32.83 (s)
	Config 2	5.21 (s)	7.84 (s)	11 (s)	15.66 (s)
	Config 3	21.43 (s)	24.76 (s)	28.77 (s)	32.85 (s)
	Config 4	12.625 (s)	16.269 (s)	19.926 (s)	24.089 (s)
	Config 5	9.825 (s)	13.449 (s)	17.616 (s)	21.789 (s)
	Config 6	3.131 (s)	6.93 (s)	10.43 (s)	14.91 (s)
80 fog nodes	Config 1	8.54 (s)	15.72 (s)	24.63 (s)	32.83 (s)
	Config 2	4.52 (s)	6.21 (s)	9.54 (s)	12.31 (s)
	Config 3	25.31 (s)	28.39 (s)	31.88 (s)	39.01 (s)
	Config 4	14.25 (s)	17.001 (s)	19.785 (s)	28.371 (s)
	Config 5	9.485 (s)	13.002 (s)	15.915 (s)	24.202 (s)
	Config 6	3.01 (s)	5.203 (s)	7.94 (s)	8.65 (s)

fog nodes increases there are fewer vehicles associated with each fog node. Accordingly, the length of a buffer is shorter and as a results the amount of time a tuple waits in a buffer until it is processed is smaller.

Table 7.16 shows the average distortion due to network latency for configuration 1, 2, 3, 4, 5 and 6 as the number of fog nodes and vehicles vary. The results show that the distortion due to network latency is zero. The reason is that in the local congestion highway notification scenario the generated tuples (position reports) represent a low volume of data.

Overall the results show that increasing the number of level 0 fog nodes results in smaller response time regardless of the configuration. If there isn't a high amount of sensor data then doing all the analysis in the cloud is better.

Table 7.13: Average End-to-End Delay-Configuration 1, 2, 3, 4, 5,and 6

Number of Level 0 Fog Nodes	Config #	500 vehicles	1000 vehicles	1500 vehicles	2000 vehicles
2 fog nodes	Config 1	0.096 (s)	0.283 (s)	0.301 (s)	0.471 (s)
	Config 2	0.0087 (s)	0.0167 (s)	0.0247 (s)	0.0327 (s)
	Config 3	0.008 (s)	0.016 (s)	0.024 (s)	0.032 (s)
	Config 4	0.008 (s)	0.015 (s)	0.023 (s)	0.03 (s)
	Config 5	0.0079 (s)	0.015 (s)	0.023 (s)	0.03 (s)
	Config 6	0.0078 (s)	0.0148 (s)	0.024 (s)	0.029 (s)
4 fog nodes	Config 1	0.096 (s)	0.283 (s)	0.301 (s)	0.471 (s)
	Config 2	0.0095 (s)	0.0095 (s)	0.0135 (s)	0.0135 (s)
	Config 3	0.004 (s)	0.008 (s)	0.012 (s)	0.016 (s)
	Config 4	0.003 (s)	0.007 (s)	0.012 (s)	0.015 (s)
	Config 5	0.0027 (s)	0.0068 (s)	0.0113 (s)	0.0142 (s)
	Config 6	0.0025 (s)	0.0061 (s)	0.0109 (s)	0.0117 (s)
10 fog nodes	Config 1	0.096 (s)	0.283 (s)	0.301 (s)	0.471 (s)
	Config 2	0.0055 (s)	0.0071 (s)	0.0087 (s)	0.0103 (s)
	Config 3	0.0018 (s)	0.0034 (s)	0.005 (s)	0.0066 (s)
	Config 4	0.0017 (s)	0.0033 (s)	0.0049 (s)	0.0065 (s)
	Config 5	0.00023 (s)	0.00327 (s)	0.00487 (s)	0.00647 (s)
	Config 6	0.00011 (s)	0.00202 (s)	0.00299 (s)	0.00401 (s)
20 fog nodes	Config 1	0.096 (s)	0.283 (s)	0.301 (s)	0.471 (s)
	Config 2	0.0012 (s)	0.0094 (s)	0.0102 (s)	0.011 (s)
	Config 3	0.0012 (s)	0.002 (s)	0.0028 (s)	0.0034 (s)
	Config 4	0.001 (s)	0.0018 (s)	0.0026 (s)	0.0034 (s)
	Config 5	0.00092 (s)	0.00172 (s)	0.00252 (s)	0.00332 (s)
	Config 6	0.00021 (s)	0.00167 (s)	0.00211 (s)	0.00297 (s)
50 fog nodes	Config 1	0.096 (s)	0.283 (s)	0.301 (s)	0.471 (s)
	Config 2	0.01982 (s)	0.01714 (s)	0.02046 (s)	0.02078 (s)
	Config 3	0.00152 (s)	0.00184 (s)	0.00216 (s)	0.00248 (s)
	Config 4	0.00092 (s)	0.00124 (s)	0.0028 (s)	0.00188 (s)
	Config 5	0.00072 (s)	0.00104 (s)	0.00136 (s)	0.00168 (s)
	Config 6	0.000481 (s)	0.00096 (s)	0.00128 (s)	0.00137 (s)
60 fog nodes	Config 1	0.096 (s)	0.283 (s)	0.301 (s)	0.471 (s)
	Config 2	0.02361 (s)	0.0238 (s)	0.024 (s)	0.024216 (s)
	Config 3	0.00141 (s)	0.00172 (s)	0.00202 (s)	0.002316 (s)
	Config 4	0.00101 (s)	0.0012 (s)	0.0012 (s)	0.001616 (s)
	Config 5	0.00071 (s)	0.0009 (s)	0.0011 (s)	0.001316 (s)
	Config 6	0.000064 (s)	0.00081 (s)	0.00082 (s)	0.00115 (s)
80 fog nodes	Config 1	0.096 (s)	0.283 (s)	0.301 (s)	0.471 (s)
	Config 2	0.03134 (s)	0.0331 (s)	0.03163 (s)	0.03184 (s)
	Config 3	0.00214 (s)	0.00288 (s)	0.00243 (s)	0.00264 (s)
	Config 4	0.00114 (s)	0.00128 (s)	0.00143 (s)	0.00164 (s)
	Config 5	0.00084 (s)	0.00098 (s)	0.00113 (s)	0.00134 (s)
	Config 6	0.000059 (s)	0.00062 (s)	0.00079 (s)	0.00109 (s)

7.6 Results of Regional Camera Surveillance Scenario

In this scenario the goal is to measure the crowd size by using cameras. One of the goals of crowd size measurement is to ensure safety and security in public roadways in the case of emergency evacuations as the result of a fire or if a natural disaster occurs [143]. Emergency evacuations require an estimate of crowd size for planning a path for evacuation path planning. Camera video streams can be used to determine the crowd size. Periodically a camera sends a tuple of data. Each tuple consist of a timestamp, camera location information as identified by longitude and latitude, camera identifier, and a sequence of video frames taken over a specific period of time. An intersection represents the minimal unit for which crowd size is calculated

Table 7.14: Average Response Time-Configuration 1, 2, 3, 4, 5,and 6

Number of Level 0 Fog Nodes	Config #	500 vehicles	1000 vehicles	1500 vehicles	2000 vehicles
2 fog nodes	Config 1	8.636 (s)	16.003 (s)	24.931 (s)	33.301 (s)
	Config 2	3068.8187 (s)	16938.6867 (s)	30806.5747 (s)	44673.0427 (s)
	Config 3	3069.338 (s)	16939.226 (s)	30808.664 (s)	44676.672 (s)
	Config 4	3069.185 (s)	16938.285 (s)	30808.018 (s)	44674.178 (s)
	Config 5	3068.6889 (s)	16936.1 (s)	30806.531 (s)	44674.048 (s)
	Config 6	3068.0178 (s)	16936.3648 (s)	30806.794 (s)	44673.809 (s)
4 fog nodes	Config 1	8.636 (s)	16.003 (s)	24.931 (s)	33.301 (s)
	Config 2	58.5395 (s)	3069.3495 (s)	10003.4535 (s)	16935.3335 (s)
	Config 3	60.424 (s)	3070.818 (s)	10004.262 (s)	16940.236 (s)
	Config 4	60.075 (s)	3070.075 (s)	10003.573 (s)	16939.037 (s)
	Config 5	59.0877 (s)	3068.8898 (s)	10001.3973 (s)	16937.9312 (s)
	Config 6	58.2125 (s)	3067.0261 (s)	10001.2709 (s)	16937.3617 (s)
10 fog nodes	Config 1	8.636 (s)	16.003 (s)	24.931 (s)	33.301 (s)
	Config 2	22.5955 (s)	45.5071 (s)	58.1187 (s)	292.5503 (s)
	Config 3	25.4118 (s)	48.1634 (s)	60.515 (s)	295.9166 (s)
	Config 4	23.752 (s)	46.252 (s)	58.902 (s)	294.912 (s)
	Config 5	22.4533 (s)	45.04827 (s)	55.46987 (s)	291.42247 (s)
	Config 6	20.61011 (s)	42.56202 (s)	54.62299 (s)	287.69401 (s)
20 fog nodes	Config 1	8.636 (s)	16.003 (s)	24.931 (s)	33.301 (s)
	Config 2	11.3512 (s)	22.3394 (s)	34.2402 (s)	45.861 (s)
	Config 3	17.9112 (s)	28.412 (s)	40.5428 (s)	52.0734 (s)
	Config 4	14.895 (s)	25.235 (s)	37.315 (s)	48.335 (s)
	Config 5	13.42192 (s)	24.47372 (s)	36.43452 (s)	47.49532 (s)
	Config 6	10.11021 (s)	19.43167 (s)	33.49211 (s)	42.02297 (s)
50 fog nodes	Config 1	8.636 (s)	16.003 (s)	24.931 (s)	33.301 (s)
	Config 2	5.59982 (s)	10.68714 (s)	14.60046 (s)	19.55078 (s)
	Config 3	16.07152 (s)	23.34184 (s)	27.24216 (s)	32.41248 (s)
	Config 4	11.585 (s)	16.117 (s)	20.755 (s)	25.384 (s)
	Config 5	9.25572 (s)	13.86504 (s)	18.42636 (s)	23.60268 (s)
	Config 6	4.940481 (s)	12.03096 (s)	16.38128 (s)	24.98137 (s)
60 fog nodes	Config 1	8.636 (s)	16.003 (s)	24.931 (s)	33.301 (s)
	Config 2	5.23361 (s)	7.24148 (s)	11.024 (s)	15.684216 (s)
	Config 3	21.43141 (s)	24.76172 (s)	28.7701 (s)	32.852316 (s)
	Config 4	12.625 (s)	16.269 (s)	19.926 (s)	24.089 (s)
	Config 5	9.82571 (s)	13.4499 (s)	17.6171 (s)	21.790316 (s)
	Config 6	3.131064 (s)	6.93081 (s)	10.43082 (s)	14.91115 (s)
80 fog nodes	Config 1	8.636 (s)	16.003 (s)	24.931 (s)	33.301 (s)
	Config 2	4.55134 (s)	6.2431 (s)	9.57163 (s)	12.34184 (s)
	Config 3	25.31214 (s)	28.39288 (s)	31.88243 (s)	39.01264 (s)
	Config 4	14.25 (s)	17.001 (s)	19.785 (s)	28.371 (s)
	Config 5	9.48584 (s)	13.00298 (s)	15.91613 (s)	24.20334 (s)
	Config 6	3.010059 (s)	5.20362 (s)	7.94079 (s)	8.65109 (s)

[88]. Intersections can be determined based on the longitude and latitude found in the tuple sent by the cameras. Each intersection is associated with a camera. A region includes multiple intersections. A region has boundaries that can be represented in terms of longitudes and latitudes. An area is defined with using four coordinates where each coordinate is associated with a longitude and latitude values.

For configuration 1 all the operators in queries Q_1, Q_2, Q_3, Q_4, Q_5, and Q_6 are placed in a cloud. For configuration 2 the query operators in queries Q_1, Q_2, Q_3, and Q_5 are mapped to level 0 fog nodes and the query operators in Q_4 and Q_6 are mapped to a cloud. For configurations 3, 4, 5, and 6 the query operators in queries Q_1, Q_2, Q_3, and Q_5 are mapped to level 0 fog nodes and the query operators in queries Q_4 and Q_6 are mapped to level 1 fog nodes.

Table 7.15: Average Distortion Due to Buffering-Configuration 1, 2, 3, 4, 5,and 6

Number of Level 0 Fog Nodes	Config #	500 vehicles	1000 vehicles	1500 vehicles	2000 vehicles
2 fog nodes	Config 1	0 (%)	0 (%)	0 (%)	0 (%)
	Config 2	45.11 (%)	48.94 (%)	49.79 (%)	49.88 (%)
	Config 3	44.11 (%)	48.04 (%)	49.22 (%)	49.57 (%)
	Config 4	43.13 (%)	46.21 (%)	49.08 (%)	49.11 (%)
	Config 5	43.09 (%)	46.13 (%)	48.11 (%)	49.02 (%)
	Config 6	43.08 (%)	46.14 (%)	48.12 (%)	49.02 (%)
4 fog nodes	Config 1	0 (%)	0 (%)	0 (%)	0 (%)
	Config 2	0 (%)	45.23 (%)	48.93 (%)	49.75 (%)
	Config 3	0 (%)	44 (%)	47.70 (%)	48.24 (%)
	Config 4	0 (%)	42.87 (%)	46.37 (%)	46.17 (%)
	Config 5	0 (%)	42.71 (%)	46.34 (%)	46.02 (%)
	Config 6	0 (%)	41.92 (%)	45.19 (%)	44.82 (%)
10 fog nodes	Config 1	0 (%)	0 (%)	0 (%)	0 (%)
	Config 2	0 (%)	0 (%)	0 (%)	0 (%)
	Config 3	0 (%)	0 (%)	0 (%)	0 (%)
	Config 4	0 (%)	0 (%)	0 (%)	0 (%)
	Config 5	0 (%)	0 (%)	0 (%)	0 (%)
	Config 6	0 (%)	0 (%)	0 (%)	0 (%)
20 fog nodes	Config 1	0 (%)	0 (%)	0 (%)	0 (%)
	Config 2	0 (%)	0 (%)	0 (%)	0 (%)
	Config 3	0 (%)	0 (%)	0 (%)	0 (%)
	Config 4	0 (%)	0 (%)	0 (%)	0 (%)
	Config 5	0 (%)	0 (%)	0 (%)	0 (%)
	Config 6	0 (%)	0 (%)	0 (%)	0 (%)
50 fog nodes	Config 1	0 (%)	0 (%)	0 (%)	0 (%)
	Config 2	0 (%)	0 (%)	0 (%)	0 (%)
	Config 3	0 (%)	0 (%)	0 (%)	0 (%)
	Config 4	0 (%)	0 (%)	0 (%)	0 (%)
	Config 5	0 (%)	0 (%)	0 (%)	0 (%)
	Config 6	0 (%)	0 (%)	0 (%)	0 (%)
60 fog nodes	Config 1	0 (%)	0 (%)	0 (%)	0 (%)
	Config 2	0 (%)	0 (%)	0 (%)	0 (%)
	Config 3	0 (%)	0 (%)	0 (%)	0 (%)
	Config 4	0 (%)	0 (%)	0 (%)	0 (%)
	Config 5	0 (%)	0 (%)	0 (%)	0 (%)
	Config 6	0 (%)	0 (%)	0 (%)	0 (%)
80 fog nodes	Config 1	0 (%)	0 (%)	0 (%)	0 (%)
	Config 2	0 (%)	0 (%)	0 (%)	0 (%)
	Config 3	0 (%)	0 (%)	0 (%)	0 (%)
	Config 4	0 (%)	0 (%)	0 (%)	0 (%)
	Config 5	0 (%)	0 (%)	0 (%)	0 (%)
	Config 6	0 (%)	0 (%)	0 (%)	0 (%)

7.6.1 Experimental Configurations

The experiment configurations for the Regional Camera Surveillance Scenario are the same as in section 7.5.

The number of level 0 fog nodes varied as follows: 2, 4, 8, 16, 32, 64. All level 0 fog nodes of a region are assigned to a level 1 fog node. The total number of cameras varies as 8, 16, 32, 64, 128, 256, and 512. In this simulations the fog nodes have the processing power 1700 MIPS for level 0 and the processing power for fog nodes in level 1 was considered to be 2700 MIPS [179]. The size of video tuples varies between 190 KB and 204 KB.

Table 7.16: Average Distortion Due to Network Latency-Configuration 1, 2, 3, 4, 5,and 6

Number of Level 0 Fog Nodes	Config #	500 vehicles	1000 vehicles	1500 vehicles	2000 vehicles
2 fog nodes	Config 1	0 (%)	0 (%)	0 (%)	0 (%)
	Config 2	0 (%)	0 (%)	0 (%)	0 (%)
	Config 3	0 (%)	0 (%)	0 (%)	0 (%)
	Config 4	0 (%)	0 (%)	0 (%)	0 (%)
	Config 5	0 (%)	0 (%)	0 (%)	0 (%)
	Config 6	0 (%)	0 (%)	0 (%)	0 (%)
4 fog nodes	Config 1	0 (%)	0 (%)	0 (%)	0 (%)
	Config 2	0 (%)	0 (%)	0 (%)	0 (%)
	Config 3	0 (%)	0 (%)	0 (%)	0 (%)
	Config 4	0 (%)	0 (%)	0 (%)	0 (%)
	Config 5	0 (%)	0 (%)	0 (%)	0 (%)
	Config 6	0 (%)	0 (%)	0 (%)	0 (%)
10 fog nodes	Config 1	0 (%)	0 (%)	0 (%)	0 (%)
	Config 2	0 (%)	0 (%)	0 (%)	0 (%)
	Config 3	0 (%)	0 (%)	0 (%)	0 (%)
	Config 4	0 (%)	0 (%)	0 (%)	0 (%)
	Config 5	0 (%)	0 (%)	0 (%)	0 (%)
	Config 6	0 (%)	0 (%)	0 (%)	0 (%)
20 fog nodes	Config 1	0 (%)	0 (%)	0 (%)	0 (%)
	Config 2	0 (%)	0 (%)	0 (%)	0 (%)
	Config 3	0 (%)	0 (%)	0 (%)	0 (%)
	Config 4	0 (%)	0 (%)	0 (%)	0 (%)
	Config 5	0 (%)	0 (%)	0 (%)	0 (%)
	Config 6	0 (%)	0 (%)	0 (%)	0 (%)
50 fog nodes	Config 1	0 (%)	0 (%)	0 (%)	0 (%)
	Config 2	0 (%)	0 (%)	0 (%)	0 (%)
	Config 3	0 (%)	0 (%)	0 (%)	0 (%)
	Config 4	0 (%)	0 (%)	0 (%)	0 (%)
	Config 5	0 (%)	0 (%)	0 (%)	0 (%)
	Config 6	0 (%)	0 (%)	0 (%)	0 (%)
60 fog nodes	Config 1	0 (%)	0 (%)	0 (%)	0 (%)
	Config 2	0 (%)	0 (%)	0 (%)	0 (%)
	Config 3	0 (%)	0 (%)	0 (%)	0 (%)
	Config 4	0 (%)	0 (%)	0 (%)	0 (%)
	Config 5	0 (%)	0 (%)	0 (%)	0 (%)
	Config 6	0 (%)	0 (%)	0 (%)	0 (%)
80 fog nodes	Config 1	0 (%)	0 (%)	0 (%)	0 (%)
	Config 2	0 (%)	0 (%)	0 (%)	0 (%)
	Config 3	0 (%)	0 (%)	0 (%)	0 (%)
	Config 4	0 (%)	0 (%)	0 (%)	0 (%)
	Config 5	0 (%)	0 (%)	0 (%)	0 (%)
	Config 6	0 (%)	0 (%)	0 (%)	0 (%)

Table 7.17: Maximum Mapping Distortion for Congested Highway Notification Scenario

Config #	mapping distortion
Config 1	0.06
Config 2	0.06
Config 3,4,5,6	0.5

7.6.2 Results for Configuration 1, 2, 3, 4, 5, and 6

From Table 7.18 we make the following observations:

- There is no significant difference for average execution time for a small set of cameras

in configurations 2, 3, 4, 5, and 6. Configuration 6 has the smallest execution time. In configurations 2, 3, 4, 5, and 6 level 0 fog nodes perform analysis on video streams and the results are produced for a level 1 fog node or cloud. Performing analysis on video streams requires high processing power which level 0 fog nodes perform in configurations 2, 3, 4, 5 and 6. For configurations 3, 4, 5 and 6 the results of video stream analysis are sent to level 1 fog nodes to measure the crowd size for a region and in configuration 2, results of video stream analysis is sent to a cloud to measure the crowd size for a region. In other words, level 1 fog nodes (configurations 3, 4, 5, 6) and cloud (configuration 2) are responsible for aggregation of results from level 0 fog nodes. Since aggregation is not computationally intensive, there is no significant difference in the average execution time for configuration 3, 4 and 5. However, by increasing the number of level 1 fog nodes, in configuration 6, we achieved lower execution time.

- Table 7.18 shows that the use of a sufficient number of fog nodes at level 0 results in a shorter execution time than using only the cloud since increasing the level 0 fog nodes meant that more computational power is introduced close to data sources.

Table 7.19 shows the average end-to-end delay for configurations 1, 2, 3, 4, 5 and 6. The results in Table 7.19 show that configuration 6 has the lowest end-to-end delay. We observed that for a small number of cameras end-to-end delay decreases by increasing the number of level 0 fog nodes. The generated video stream by cameras is distributed among level 0 fog nodes and thus communication networks are not overwhelmed with video stream traffic. However, for a large number of cameras end-to-end delay decreases by increasing the number of level 0 fog nodes. The generated data by level 0 fog nodes needs to be transmitted to a level 1 fog node.

Table 7.20 shows the average response time for configurations 1, 2, 3, 4, 5 and 6. Table 7.20 shows that configuration 6 results in a better response time. Also, configuration 6 has the lowest response time for a large number of cameras. The results show the benefits of placing computing resource close the data sources when the number of cameras increases.

Table 7.21 shows the average distortion due to buffering for configurations 1, 2, 3, 4, 5 and 6. The results show that increasing the number of fog nodes provides better quality of service. Table 7.21 shows that as the number of fog nodes increases the distortion decreases. Increasing the number of fog nodes means more available processing resources close to the data sources. A data stream tuple (video tuple) is placed in a fog node's buffer upon its arrival. The fog node reads from the buffer and processes the tuples. As the number of fog nodes increases there are fewer cameras associated with each fog node. This means that the buffer is smaller as the number of level 0 fog nodes increases and thus the amount of time that a tuple waits in the buffer is smaller.

Table 7.22 shows average distortion as the number of fog nodes and cameras vary. We can make the following conclusions:

- Table 7.22 shows that by increasing the number of level 0 fog nodes we can better cope with the network latency for a large number of data sources by increasing the number of fog nodes. Decreasing in end-to-end delay decreases the average distortion.

- Table 7.22 shows that for a large number of cameras end-to-end delay decreases by increasing the number of level 0 fog nodes. However, increasing the number of level

Table 7.18: Average Execution Time-Configuration 1, 2, 3, 4, 5, and 6

# of Level 0 Fog Nodes	Config #	8 Cameras	16 Cameras	32 Cameras	64 Cameras	128 Cameras	256 Cameras	512 Cameras
2 fog nodes	Config 1	8.42(s)	17.05 (s)	33.53(s)	62.84(s)	1035.46(s)	3842.37(s)	14985.23 (s)
	Config 2	684.63 (s)	3174.26 (s)	8165.73 (s)	18179.53 (s)	38137.24(s)	78141.55 (s)	161751.81 (s)
	Config 3	699.91 (s)	3198.23 (s)	8196.42 (s)	18202.33 (s)	38217.72(s)	78231.13 (s)	161938.11 (s)
	Config 4	699.522 (s)	3197.452 (s)	8194.868 (s)	18199.166 (s)	38211.512(s)	78218.714 (s)	161911.73 (s)
	Config 5	699.108 (s)	3197.11 (s)	8192.416 (s)	18192.043 (s)	38192.218(s)	78173.151 (s)	161808.11 (s)
	Config 6	699.05 (s)	3197.03 (s)	8192.5 (s)	18192.33 (s)	38192.104(s)	78173.16 (s)	161808.52 (s)
4 fog nodes	Config 1	8.42(s)	17.05 (s)	33.53(s)	62.84(s)	1035.46(s)	3842.37(s)	14985.23 (s)
	Config 2	55.37 (s)	689.74 (s)	3178 (s)	8166 (s)	18182.74 (s)	38134.64(s)	78557.35 (s)
	Config 3	59.13 (s)	694.55 (s)	3195.23 (s)	8195.81 (s)	18199.21 (s)	38212.98(s)	79865.12(s)
	Config 4	58.936 (s)	694.161 (s)	3194.454 (s)	8194.228 (s)	18196.106 (s)	38206.772 (s)	78705.95 (s)
	Config 5	58.742 (s)	693.772 (s)	3193.678 (s)	8192.646 (s)	18193.002 (s)	38200.564 (s)	78693.16 (s)
	Config 6	54.2 (s)	681.12 (s)	3181.811 (s)	8187.011 (s)	18180.23 (s)	38182.32 (s)	78581.32 (s)
8 fog nodes	Config 1	8.42(s)	17.05 (s)	33.53(s)	62.84(s)	1035.46(s)	3842.37(s)	14985.23 (s)
	Config 2	26.76 (s)	56.46 (s)	686.46 (s)	3172.63 (s)	8162.82 (s)	18182.54 (s)	40001.58 (s)
	Config 3	30.42 (s)	59.39 (s)	694.19 (s)	3188.33 (s)	8194.91 (s)	18201.34(s)	40042.94 (s)
	Config 4	30.129 (s)	58.808 (s)	693.026 (s)	3186.002 (s)	8190.26 (s)	18192.028(s)	40023.46 (s)
	Config 5	30.032 (s)	58.602 (s)	692.638 (s)	3185.162 (s)	8188.702 (s)	18188.924 (s)	40006.63 (s)
	Config 6	20.07 (s)	20.37 (s)	679.11 (s)	3122.92 (s)	8131.34 (s)	18126.94 (s)	39016.82 (s)
16 fog nodes	Config 1	8.42(s)	17.05 (s)	33.53(s)	62.84(s)	1035.46(s)	3842.37(s)	14985.23 (s)
	Config 2	26.73 (s)	26.84 (s)	55.61 (s)	687.96 (s)	3178.84 (s)	8166.19 (s)	16904.01 (s)
	Config 3	29.38 (s)	29.11 (s)	58.83 (s)	689.334 (s)	3196.41(s)	8183.45 (s)	17185.24 (s)
	Config 4	28.856 (s)	28.586 (s)	57.79 (s)	687.27 (s)	3192.21(s)	8175.06 (s)	17004.12 (s)
	Config 5	28.602 (s)	28.333 (s)	57.278 (s)	686.226 (s)	3190.201(s)	8171.034 (s)	16914.04 (s)
	Config 6	17.47 (s)	17.73 (s)	17.81 (s)	661.38(s)	3002.87 (s)	3911.09 (s)	15821.44 (s)
32 fog nodes	Config 1	8.42(s)	17.05 (s)	33.53(s)	62.84(s)	1035.46(s)	3842.37(s)	14985.23 (s)
	Config 2	26.25(s)	26.32 (s)	25.98 (s)	56.34 (s)	685.28 (s)	3172.36	6661.956 (s)
	Config 3	29.82(s)	29.32 (s)	28.31 (s)	58.42 (s)	687.42(s)	3185.51(s)	7645.224 (s)
	Config 4	28.85(s)	28.35 (s)	31.97 (s)	56.48 (s)	683.52(s)	3173.81(s)	6982.382 (s)
	Config 5	28.26(s)	27.76 (s)	26.75 (s)	55.32 (s)	681.22(s)	3173.01(s)	6618.719 (s)
	Config 6	14.65(s)	14.81 (s)	16.21 (s)	20.82 (s)	648.12 (s)	3068.51 (s)	3815.31 (s)
60 fog nodes	Config 1	8.42(s)	17.05 (s)	33.53(s)	62.84(s)	1035.46(s)	3842.37(s)	14985.23 (s)
	Config 2	26.21(s)	25.92 (s)	25.93 (s)	29.37 (s)	61.52 (s)	696.57 (s)	2236.52 (s)
	Config 3	27.93(s)	28.31 (s)	27.31 (s)	28.21 (s)	57.92 (s)	688.41 (s)	2301.57 (s)
	Config 4	27.18(s)	25.09 (s)	25.82 (s)	26.31 (s)	54.92 (s)	681.32 (s)	2224.13 (s)
	Config 5	23.21(s)	24.67 (s)	24.72 (s)	24.28 (s)	50.64 (s)	626.52 (s)	2092.16(s)
	Config 6	11.13 (s)	11.56 (s)	12.83 (s)	14.25 (s)	17.97 (s)	511.93 (s)	1937.13 (s)
64 fog nodes	Config 1	8.42(s)	17.05 (s)	33.53(s)	62.84(s)	1035.46(s)	3842.37(s)	14985.23 (s)
	Config 2	25.88(s)	25.27 (s)	25.33 (s)	26.14 (s)	55.73 (s)	681.22 (s)	2201.13 (s)
	Config 3	28.43(s)	28.89 (s)	28.44 (s)	28.92 (s)	59.71 (s)	691.33 (s)	2313.52 (s)
	Config 4	26.4(s)	26.86 (s)	26.41 (s)	26.89 (s)	55.68 (s)	683.21 (s)	2264.43 (s)
	Config 5	25.33(s)	25.79 (s)	25.34 (s)	25.82 (s)	53.51 (s)	678.93 (s)	2186.28(s)
	Config 6	12.52 (s)	12.77 (s)	13.02 (s)	14.83 (s)	18.7 (s)	572.63 (s)	1386.84 (s)
70 fog nodes	Config 1	8.42(s)	17.05 (s)	33.53(s)	62.84(s)	1035.46(s)	3842.37(s)	14985.23 (s)
	Config 2	23.27(s)	23.73 (s)	23.81 (s)	24.09 (s)	51.37 (s)	648.32 (s)	2163.52 (s)
	Config 3	29.61(s)	29.29 (s)	29.54 (s)	30.06 (s)	61.11 (s)	694.91 (s)	2324.92 (s)
	Config 4	27.34(s)	27.3 (s)	27.64 (s)	28.32 (s)	56.35 (s)	691.64 (s)	2271.40 (s)
	Config 5	26.41(s)	26.09 (s)	26.94 (s)	27.14 (s)	55.14 (s)	698.98 (s)	2223.71(s)
	Config 6	13.91 (s)	13.08 (s)	13.97 (s)	15.17 (s)	21.83 (s)	772.02 (s)	1517.64 (s)
80 fog nodes	Config 1	8.42(s)	17.05 (s)	33.53(s)	62.84(s)	1035.46(s)	3842.37(s)	14985.23 (s)
	Config 2	22.21(s)	23.77 (s)	23.84 (s)	24.39 (s)	48.95 (s)	634.23 (s)	2113.22 (s)
	Config 3	28.93(s)	29.41 (s)	29.83 (s)	29.15 (s)	59.51 (s)	692.03 (s)	2321.63 (s)
	Config 4	28.02(s)	27.93 (s)	27.55 (s)	28.27 (s)	54.25 (s)	691.44 (s)	2267.53 (s)
	Config 5	26.34(s)	25.89 (s)	27.02 (s)	26.02 (s)	54.24 (s)	695.16 (s)	2219.43(s)
	Config 6	13.42 (s)	13.11 (s)	13.12 (s)	15.21 (s)	19.62 (s)	768.93 (s)	1509.71 (s)

0 fog nodes will lead to an increase in end-to-end delay and increasing the end-to-end delay results in an increase of the average distortion due to network latency.

- Table 7.22 shows the average distortion due to network latency for configuration 1, 2, 3, 4, 5 and 6. As Table 7.22 shows the average distortion due to network latency decreases as the number of level 0 fog nodes increased for a small set of level 0 fog nodes. However the average distortion due to network latency increases for a large number of level 0 fog nodes since more data is generated by increasing the level 0 fog nodes. Moreover, there

Table 7.19: Average End-to-End Delay-Configuration 1, 2, 3, 4, 5, and 6

# of Level 0 Fog Nodes	Config #	8 Cameras	16 Cameras	32 Cameras	64 Cameras	128 Cameras	256 Cameras	512 Cameras
2 fog nodes	Config 1	0.5988(s)	1.167 (s)	2.5612(s)	4.7099 (s)	9.1316(s)	19.8418 (s)	44.243 (s)
	Config 2	0.0613 (s)	0.1305 (s)	0.266 (s)	0.5328 (s)	1.235(s)	2.229(s)	6.725 (s)
	Config 3	0.057 (s)	0.1155 (s)	0.232 (s)	0.4525 (s)	0.966(s)	1.846(s)	4.215 (s)
	Config 4	0.0565 (s)	0.1137 (s)	0.228 (s)	0.456 (s)	0.931(s)	1.831(s)	4.206 (s)
	Config 5	0.0563 (s)	0.1131 (s)	0.2266 (s)	0.4533 (s)	0.9193(s)	1.8183(s)	4.181 (s)
	Config 6	0.0564 (s)	0.113 (s)	0.2264 (s)	0.4531 (s)	0.9195 (s)	1.8184 (s)	4.1808 (s)
4 fog nodes	Config 1	0.5988(s)	1.167 (s)	2.5612(s)	4.7099 (s)	9.1316(s)	19.8418 (s)	44.243 (s)
	Config 2	0.0596 (s)	0.0769 (s)	0.1542 (s)	0.3054 (s)	0.6176(s)	1.2363(s)	5.633(s)
	Config 3	0.0311 (s)	0.061 (s)	0.117 (s)	0.288 (s)	0.498(s)	0.998 (s)	3.187 (s)
	Config 4	0.029 (s)	0.0585 (s)	0.1145 (s)	0.234 (s)	0.468 (s)	0.936 (s)	3.109 (s)
	Config 5	0.0286 (s)	0.0576 (s)	0.1136 (s)	0.23 (s)	0.461 (s)	0.922 (s)	3.097 (s)
	Config 6	0.0272 (s)	0.0553 (s)	0.111 (s)	0.213 (s)	0.428 (s)	0.909 (s)	3.002 (s)
8 fog nodes	Config 1	0.5988(s)	1.167 (s)	2.5612(s)	4.7099 (s)	9.1316(s)	19.8418 (s)	44.243 (s)
	Config 2	0.0564 (s)	0.1117 (s)	0.2261 (s)	0.4528 (s)	0.9024 (s)	1.7051 (s)	5.648 (s)
	Config 3	0.022 (s)	0.0451 (s)	0.089 (s)	0.1781 (s)	0.355 (s)	0.7034(s)	2.822 (s)
	Config 4	0.0188 (s)	0.036 (s)	0.072 (s)	0.144 (s)	0.281 (s)	0.576(s)	2.673 (s)
	Config 5	0.0168 (s)	0.033 (s)	0.067 (s)	0.133 (s)	0.0.262 (s)	0.5341(s)	2.902(s)
	Config 6	0.0132 (s)	0.0281 (s)	0.049 (s)	0.125 (s)	0.247 (s)	0.528(s)	2.814(s)
16 fog nodes	Config 1	0.5988(s)	1.167 (s)	2.5612(s)	4.7099 (s)	9.1316(s)	19.8418 (s)	44.243 (s)
	Config 2	0.0541 (s)	0.1833 (s)	0.3688(s)	0.7282 (s)	1.4673 (s)	2.8365(s)	8.0417(s)
	Config 3	0.0218 (s)	0.045 (s)	0.093 (s)	0.1834 (s)	0.3692 (s)	0.7374(s)	2.872 (s)
	Config 4	0.0183 (s)	0.037 (s)	0.067 (s)	0.124 (s)	0.242 (s)	0.4834(s)	2.565 (s)
	Config 5	0.0164 (s)	0.024 (s)	0.049 (s)	0.0986 (s)	0.1974 (s)	0.3946(s)	2.4577 (s)
	Config 6	0.0123 (s)	0.0212 (s)	0.0453 (s)	0.0939 (s)	0.1927 (s)	0.3911(s)	2.4515 (s)
32 fog nodes	Config 1	0.5988(s)	1.167 (s)	2.5612(s)	4.7099 (s)	9.1316(s)	19.8418 (s)	44.243 (s)
	Config 2	0.0558 (s)	0.1853 (s)	0.6924 (s)	2.3926 (s)	2.7764 (s)	5.5442(s)	13.864 (s)
	Config 3	0.0206 (s)	0.0458 (s)	0.142(s)	0.5911 (s)	0.7721 (s)	1.1409 (s)	3.2892 (s)
	Config 4	0.0186 (s)	0.0322 (s)	0.0786(s)	0.156 (s)	0.312 (s)	0.618 (s)	2.7045(s)
	Config 5	0.0163 (s)	0.02456 (s)	0.056 (s)	0.1133 (s)	0.2213 (s)	0.4571 (s)	2.5117 (s)
	Config 6	0.0148 (s)	0.0236 (s)	0.0524 (s)	0.108 (s)	0.2201 (s)	0.4482 (s)	2.509 (s)
60 fog nodes	Config 1	0.5988(s)	1.167 (s)	2.5612(s)	4.7099 (s)	9.1316(s)	19.8418 (s)	44.243 (s)
	Config 2	0.0558 (s)	0.1852 (s)	0.6895 (s)	2.7452 (s)	5.4847 (s)	11.9716 (s)	15.5221 (s)
	Config 3	0.0207 (s)	0.0449 (s)	0.1389 (s)	0.5448 (s)	1.0726 (s)	2.1533 (s)	4.9053 (s)
	Config 4	0.0189 (s)	0.0319 (s)	0.0785 (s)	0.2852 (s)	0.569 (s)	0.8874 (s)	4.651 (s)
	Config 5	0.0165 (s)	0.0247 (s)	0.0492 (s)	0.1992 (s)	0.3411 (s)	0.6247 (s)	2.6932 (s)
	Config 6	0.0149 (s)	0.0231 (s)	0.0526 (s)	0.1961 (s)	0.3917 (s)	0.6241 (s)	3.9816 (s)
64 fog nodes	Config 1	0.5988(s)	1.167 (s)	2.5612(s)	4.7099 (s)	9.1316(s)	19.8418 (s)	44.243 (s)
	Config 2	0.0562 (s)	0.1859 (s)	0.6893 (s)	2.7447 (s)	5.4844 (s)	10.9711 (s)	14.726 (s)
	Config 3	0.0201 (s)	0.0445 (s)	0.1382 (s)	0.5441 (s)	1.0723 (s)	2.1528 (s)	4.454 (s)
	Config 4	0.0184 (s)	0.0311 (s)	0.0781 (s)	0.284 (s)	0.562 (s)	0.886 (s)	3.01 (s)
	Config 5	0.0161 (s)	0.02413 (s)	0.0488 (s)	0.1986 (s)	0.393 (s)	0.6241 (s)	2.711 (s)
	Config 6	0.0148 (s)	0.0229 (s)	0.0525 (s)	0.195 (s)	0.3911 (s)	0.623 (s)	2.706 (s)
70 fog nodes	Config 1	0.5988(s)	1.167 (s)	2.5612(s)	4.7099 (s)	9.1316(s)	19.8418 (s)	44.243 (s)
	Config 2	0.0561 (s)	0.1851 (s)	0.6891 (s)	2.7421 (s)	5.4831 (s)	10.9688 (s)	14.7252 (s)
	Config 3	0.0202 (s)	0.0443 (s)	0.1383 (s)	0.5432 (s)	1.0704 (s)	2.1511 (s)	4.4517 (s)
	Config 4	0.0181 (s)	0.0324 (s)	0.0782 (s)	0.2831 (s)	0.5614 (s)	0.8828 (s)	2.9021 (s)
	Config 5	0.0159 (s)	0.0246 (s)	0.0474 (s)	0.1956 (s)	0.3919 (s)	0.6212 (s)	2.6893 (s)
	Config 6	0.0133 (s)	0.0228 (s)	0.0523 (s)	0.1937 (s)	0.3893 (s)	0.6214 (s)	2.6849 (s)
80 fog nodes	Config 1	0.5988(s)	1.167 (s)	2.5612(s)	4.7099 (s)	9.1316(s)	19.8418 (s)	44.243 (s)
	Config 2	0.0559 (s)	0.1849 (s)	0.6907 (s)	2.7419 (s)	49809 (s)	8.6327 (s)	12.9802 (s)
	Config 3	0.0201 (s)	0.0441 (s)	0.1388 (s)	0.5433 (s)	1.0594 (s)	2.1232 (s)	4.1646 (s)
	Config 4	0.0179 (s)	0.0327 (s)	0.0786 (s)	0.2833 (s)	0.5434 (s)	0.8755(s)	2.1943 (s)
	Config 5	0.0163 (s)	0.0244 (s)	0.0471 (s)	0.1955 (s)	0.3522 (s)	0.6128 (s)	2.1902 (s)
	Config 6	0.0137 (s)	0.0232 (s)	0.0519 (s)	0.1936 (s)	0.3508 (s)	0.6119 (s)	2.1822 (s)

is no significant difference in distortion due to network latency for a large number of cameras.

Table 7.20: Average Response Time-Configuration 1, 2, 3, 4, 5, and 6

# of Level 0 Fog Nodes	Config #	8 Cameras	16 Cameras	32 Cameras	64 Cameras	128 Cameras	256 Cameras	512 Cameras
2 fog nodes	Config 1	9.018 (s)	18.217 (s)	36.091 (s)	67.549 (s)	1044.591 (s)	3862.211 (s)	15029.473 (s)
	Config 2	684.6913 (s)	3174.3905(s)	8165.996 (s)	18180.0628 (s)	38138.475 (s)	78143.779 (s)	161758.535 (s)
	Config 3	699.967 (s)	3198.345 (s)	8196.652 (s)	18202.782 (s)	38218.172(s)	78232.976 (s)	161942.325 (s)
	Config 4	699.5785 (s)	3197.5657 (s)	8195.096 (s)	18199.622 (s)	38212.443 (s)	78220.545 (s)	161915.936 (s)
	Config 5	699.1643 (s)	3197.2231 (s)	8192.6426 (s)	18192.4963 (s)	38193.1373(s)	78174.9693 (s)	161812.291 (s)
	Config 6	699.1064 (s)	3197.143 (s)	8192.7264 (s)	18192.7831 (s)	38193.0235 (s)	78174.9784 (s)	161812.7008 (s)
4 fog nodes	Config 1	9.018 (s)	18.217 (s)	36.091 (s)	67.549 (s)	1044.591 (s)	3862.211 (s)	15029.473 (s)
	Config 2	55.4296 (s)	689.8169 (s)	3178.1542 (s)	8166.3054 (s)	18183.3576 (s)	38135.8763 (s)	78562.983 (s)
	Config 3	59.161 (s)	694.611 (s)	3195.347 (s)	8196.098 (s)	18199.708 (s)	38213.978(s)	79868.307 (s)
	Config 4	58.965 (s)	694.2195 (s)	3194.5685 (s)	8194.462 (s)	18196.574 (s)	38207.708 (s)	78709.059 (s)
	Config 5	58.7706 (s)	693.8296 (s)	3193.7916 (s)	8192.876 (s)	18193.463 (s)	38201.486 (s)	78696.257(s)
	Config 6	54.2272 (s)	681.1753 (s)	3181.922 (s)	8187.224 (s)	18180.658 (s)	38183.229 (s)	78584.322 (s)
8 fog nodes	Config 1	9.018 (s)	18.217 (s)	36.091 (s)	67.549 (s)	1044.591 (s)	3862.211 (s)	15029.473 (s)
	Config 2	26.8164 (s)	56.5717 (s)	686.6861 (s)	3173.0828 (s)	8163.7224 (s)	18184.2451 (s)	40007.228 (s)
	Config 3	30.442 (s)	59.4351 (s)	694.279 (s)	3188.508 (s)	8195.265 (s)	18202.043(s)	40045.762 (s)
	Config 4	30.1478 (s)	58.844 (s)	693.098 (s)	3186.146 (s)	8190.541 (s)	18192.604 (s)	40026.133 (s)
	Config 5	30.0488 (s)	58.635 (s)	692.705 (s)	3185.295 (s)	8188.964 (s)	18189.4581 (s)	40009.532 (s)
	Config 6	20.0832 (s)	20.3981 (s)	679.159 (s)	3123.045 (s)	8131.587 (s)	18127.468 (s)	39019.634 (s)
16 fog nodes	Config 1	9.018 (s)	18.217 (s)	36.091 (s)	67.549 (s)	1044.591 (s)	3862.211 (s)	15029.473 (s)
	Config 2	26.7841 (s)	27.0233 (s)	55.9788 (s)	688.6882 (s)	3180.3073 (s)	8169.0265 (s)	16912.0517 (s)
	Config 3	29.401 (s)	29.155 (s)	58.923 (s)	689.513 (s)	3196.779(s)	8184.187 (s)	17188.112 (s)
	Config 4	28.8743 (s)	28.623 (s)	57.857 (s)	687.394 (s)	3192.452 (s)	8175.5434 (s)	17006.685 (s)
	Config 5	28.6184 (s)	28.357 (s)	57.327 (s)	686.3246 (s)	3190.3984(s)	8171.4286 (s)	16916.4977 (s)
	Config 6	17.4823 (s)	17.7512 (s)	17.8553 (s)	661.4739 (s)	3003.0627 (s)	3911.4811 (s)	15823.8915 (s)
32 fog nodes	Config 1	9.018 (s)	18.217 (s)	36.091 (s)	67.549 (s)	1044.591 (s)	3862.211 (s)	15029.473 (s)
	Config 2	26.3058 (s)	26.5053 (s)	26.6724 (s)	58.7326 (s)	688.0564 (s)	3177.9042 (s)	6675.82 (s)
	Config 3	29.841(s)	29.365 (s)	28.452 (s)	61.711 (s)	687.992(s)	3186.650(s)	7648.5132 (s)
	Config 4	28.8686(s)	28.3822 (s)	32.0486 (s)	56.636 (s)	683.832(s)	3174.428(s)	6985.0865 (s)
	Config 5	28.2763 (s)	27.78456 (s)	26.806 (s)	55.4333 (s)	681.4413(s)	3173.4671(s)	6621.2307 (s)
	Config 6	14.6618 (s)	14.8296 (s)	16.2524 (s)	20.928 (s)	648.3401 (s)	3068.9582 (s)	3817.819 (s)
60 fog nodes	Config 1	9.018 (s)	18.217 (s)	36.091 (s)	67.549 (s)	1044.591 (s)	3862.211 (s)	15029.473 (s)
	Config 2	26.2658(s)	26.1052 (s)	26.6195 (s)	32.1152 (s)	67.0047(s)	708.5416(s)	2252.0421 (s)
	Config 3	27.9507(s)	28.3549 (s)	27.4489 (s)	28.7548 (s)	58.9926 (s)	690.5633 (s)	2306.4753 (s)
	Config 4	27.1989 (s)	25.1219 (s)	25.8985 (s)	26.5952(s)	55.489 (s)	682.2074 (s)	2228.781 (s)
	Config 5	23.2265 (s)	24.6947(s)	24.7692(s)	24.479(s)	50.9811 (s)	627.1447 (s)	2094.853 (s)
	Config 6	11.1449 (s)	11.5831 (s)	12.8826 (s)	14.4461(s)	18.3617 (s)	512.5541 (s)	1941.1116 (s)
64 fog nodes	Config 1	9.018 (s)	18.217 (s)	36.091 (s)	67.549 (s)	1044.591 (s)	3862.211 (s)	15029.473 (s)
	Config 2	25.9362(s)	25.4559 (s)	26.0193 (s)	28.8847 (s)	61.2144 (s)	692.1911 (s)	2215.856 (s)
	Config 3	28.450 (s)	28.934 (s)	28.5782 (s)	29.464 (s)	60.782 (s)	693.482 (s)	2317.974 (s)
	Config 4	26.4184 (s)	26.8911 (s)	26.4881 (s)	27.174 (s)	56.242 (s)	684.096 (s)	2267.44 (s)
	Config 5	25.3461(s)	25.81413 (s)	25.3588 (s)	26.0186 (s)	53.903 (s)	679.5541 (s)	2188.991(s)
	Config 6	12.5348 (s)	12.7929 (s)	13.0375 (s)	15.025 (s)	19.0911 (s)	573.253 (s)	1389.546 (s)
70 fog nodes	Config 1	9.018 (s)	18.217 (s)	36.091 (s)	67.549 (s)	1044.591 (s)	3862.211 (s)	15029.473 (s)
	Config 2	23.3261(s)	23.9151(s)	24.4991 (s)	26.832 (s)	56.8531 (s)	659.2888 (s)	2178.2452 (s)
	Config 3	29.6302(s)	29.3343 (s)	29.6783 (s)	30.6032 (s)	62.1804 (s)	697.0611 (s)	2329.3717 (s)
	Config 4	27.3581 (s)	27.3324 (s)	27.7182 (s)	28.6031 (s)	56.9114(s)	692.5228 (s)	2274.3021 (s)
	Config 5	26.4259 (s)	26.1146 (s)	26.987 (s)	27.3356 (s)	55.5319 (s)	699.6012 (s)	2226.3993 (s)
	Config 6	13.9233 (s)	13.1028 (s)	14.0223 (s)	15.3637 (s)	22.2193 (s)	772.6414 (s)	1520.3249 (s)
80 fog nodes	Config 1	9.018 (s)	18.217 (s)	36.091 (s)	67.549 (s)	1044.591 (s)	3862.211 (s)	15029.473 (s)
	Config 2	22.2659(s)	23.9549(s)	24.5307 (s)	24.5307 (s)	49857.95 (s)	642.8627 (s)	2126.2002(s)
	Config 3	28.9501(s)	29.4541(s)	29.9688 (s)	29.6933 (s)	60.5694 (s)	694.1532 (s)	2325.7946 (s)
	Config 4	28.0379(s)	27.9627 (s)	27.6286 (s)	28.5533 (s)	54.7934 (s)	692.3155(s)	2269.7243(s)
	Config 5	26.3563(s)	25.9144 (s)	27.0671(s)	26.2155 (s)	54.5922 (s)	695.7728(s)	2221.6202 (s)
	Config 6	13.4337 (s)	13.1332 (s)	13.1719 (s)	15.4036 (s)	19.9708 (s)	769.5419 (s)	1511.8922 (s)

Table 7.21: Average Distortion Due to Buffering-Configuration 1, 2, 3, 4, 5, and 6

# of Level 0 Fog Nodes	Config #	8 Cameras	16 Cameras	32 Cameras	64 Cameras	128 Cameras	256 Cameras	512 Cameras
2 fog nodes	Config 1	0 (%)	0 (%)	0 (%)	0 (%)	21.01 (%)	42.32 (%)	97.59 (%)
	Config 2	6.14 (%)	40.54 (%)	46.32 (%)	48.34 (%)	49.42 (%)	49.61(%)	49.88 (%)
	Config 3	7.08 (%)	40.61 (%)	46.13 (%)	48.35 (%)	49.35(%)	49.61(%)	49.88 (%)
	Config 4	7.08 (%)	40.61 (%)	46.33 (%)	48.35 (%)	49.21 (%)	49.61 (%)	49.89(%)
	Config 5	7.01 (%)	40.11 (%)	45.69 (%)	47.9 (%)	48.15 (%)	49.21 (%)	49.88 (%)
	Config 6	7.01(%)	40.1 (%)	45.69 (%)	47.89 (%)	48.15 (%)	49.19(%)	49.88 (%)
4 fog nodes	Config 1	0 (%)	0 (%)	0 (%)	0 (%)	21.01 (%)	42.32 (%)	48.79 (%)
	Config 2	0 (%)	6.12 (%)	41.09 (%)	46.29 (%)	48.13 (%)	49.23 (%)	49.77 (%)
	Config 3	0 (%)	7.12 (%)	40.21 (%)	46.41 (%)	48.67 (%)	49.11(%)	49.77 (%)
	Config 4	0 (%)	6.77 (%)	40.6 (%)	46.01 (%)	48.31 (%)	49.21 (%)	49.77 (%)
	Config 5	0 (%)	6.62 (%)	40.42 (%)	45.21 (%)	48.26 (%)	49.01 (%)	49.54 (%)
	Config 6	0(%)	6.21 (%)	39.26(%)	44.12(%)	47.82(%)	48.29(%)	49.76 (%)
8 fog nodes	Config 1	0 (%)	0 (%)	0 (%)	0 (%)	21.01 (%)	42.32 (%)	48.79 (%)
	Config 2	0 (%)	0 (%)	6.09 (%)	40.04 (%)	46.82 (%)	48.94 (%)	49.55 (%)
	Config 3	0 (%)	0 (%)	7.07 (%)	40.49 (%)	46.23 (%)	48.21 (%)	49.55 (%)
	Config 4	0 (%)	0 (%)	6.64 (%)	40.58 (%)	46.33 (%)	49.17 (%)	49.55 (%)
	Config 5	0 (%)	0 (%)	6.31 (%)	39.81 (%)	44.31 (%)	47.39 (%)	49.55(%)
	Config 6	0(%)	0 (%)	6.08 (%)	38.47 (%)	41.57 (%)	42.01 (%)	49.53 (%)
16 fog nodes	Config 1	0 (%)	0 (%)	0 (%)	0 (%)	21.01 (%)	42.32 (%)	48.79(%)
	Config 2	0 (%)	0 (%)	0 (%)	7.01 (%)	40.54 (%)	47.59 (%)	48.93 (%)
	Config 3	0 (%)	0 (%)	0 (%)	7.02 (%)	40.58 (%)	46.13 (%)	48.95 (%)
	Config 4	0 (%)	0 (%)	0 (%)	6.26 (%)	40.59 (%)	46.32 (%)	48.94 (%)
	Config 5	0 (%)	0 (%)	0 (%)	6.13 (%)	38.51 (%)	45.82 (%)	48.94 (%)
	Config 6	0(%)	0 (%)	0 (%)	0 (%)	37.56 (%)	43.01(%)	48.86 (%)
32 fog nodes	Config 1	0 (%)	0 (%)	0 (%)	0 (%)	21.01 (%)	42.32 (%)	48.79 (%)
	Config 2	0 (%)	0 (%)	0 (%)	0 (%)	6.89 (%)	41.09 (%)	47.29 (%)
	Config 3	0 (%)	0 (%)	0 (%)	0 (%)	7.09 (%)	40.52 (%)	47.64 (%)
	Config 4	0 (%)	0 (%)	0 (%)	0 (%)	5.94 (%)	40.54 (%)	47.42 (%)
	Config 5	0 (%)	0 (%)	0 (%)	0 (%)	5.94 (%)	40.54 (%)	47.28 (%)
	Config 6	0(%)	0 (%)	0 (%)	0 (%)	4.45(%)	33.26 (%)	45.28 (%)
64 fog nodes	Config 1	0 (%)	0 (%)	0 (%)	0 (%)	21.01 (%)	42.32 (%)	48.79(%)
	Config 2	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	6.74 (%)	41.82 (%)
	Config 3	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	7.04 (%)	42.1 (%)
	Config 4	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	5.75 (%)	42.05 (%)
	Config 5	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	5.52 (%)	41.76 (%)
	Config 6	0(%)	0 (%)	0(%)	0 (%)	0 (%)	4.13 (%)	27.02 (%)

Table 7.22: Average Distortion Due to Network Latency-Configuration 1, 2, 3, 4, 5, and 6

# of Level 0 Fog Nodes	Config #	8 Cameras	16 Cameras	32 Cameras	64 Cameras	128 Cameras	256 Cameras	512 Cameras
2 fog nodes	Config 1	48.78(%)	49.37 (%)	49.71(%)	49.84 (%)	49.87(%)	49.93(%)	52.03 (%)
	Config 2	38.17 (%)	44.44 (%)	47.27 (%)	48.63 (%)	49.41(%)	49.67	50.22 (%)
	Config 3	37.41 (%)	43.57 (%)	46.72 (%)	48.06 (%)	49.24 (%)	49.89 (%)	50.32 (%)
	Config 4	24.33 (%)	37.24 (%)	43.64 (%)	46.82 (%)	48.44 (%)	49.2 (%)	50.35 (%)
	Config 5	24.24 (%)	37.17 (%)	43.6 (%)	46.8 (%)	48.42 (%)	49.2 (%)	50.34 (%)
	Config 6	24.34 (%)	37.25 (%)	43.64 (%)	46.82 (%)	48.44 (%)	49.19 (%)	50.34 (%)
4 fog nodes	Config 1	48.78(%)	49.37 (%)	49.71(%)	49.84 (%)	49.87(%)	49.93(%)	52.03 (%)
	Config 2	37.83 (%)	40.72 (%)	45.29 (%)	47.62 (%)	48.82 (%)	49.41(%)	50.05 (%)
	Config 3	26.52 (%)	38.97 (%)	43.54 (%)	47.22 (%)	48.44(%)	49.21(%)	49.54 (%)
	Config 4	0 (%)	25.21 (%)	37.33 (%)	43.8 (%)	46.9(%)	48.45 (%)	49.54 (%)
	Config 5	0 (%)	24.82 (%)	37.23 (%)	43.69 (%)	46.85 (%)	48.42 (%)	49.43 (%)
	Config 6	0 (%)	24.79 (%)	37.23 (%)	43.66 (%)	46.79 (%)	48.41 (%)	49.22 (%)
8 fog nodes	Config 1	48.78(%)	49.37 (%)	49.71(%)	49.84 (%)	49.87(%)	49.93(%)	52.03 (%)
	Config 2	37.14 (%)	43.50 (%)	46.79 (%)	48.39 (%)	49.19 (%)	49.57(%)	50.05 (%)
	Config 3	17.83 (%)	33.82 (%)	41.03 (%)	46.94 (%)	47.31 (%)	48.63(%)	49.38 (%)
	Config 4	0 (%)	9.72 (%)	29.88 (%)	39.86 (%)	44.83 (%)	47.48 (%)	49.38 (%)
	Config 5	0 (%)	6.06 (%)	28.35 (%)	39.09 (%)	44.46 (%)	47.28 (%)	49.41 (%)
	Config 6	0 (%)	0 (%)	26.12 (%)	37.56 (%)	43.07 (%)	46.11 (%)	49.42 (%)
16 fog nodes	Config 1	48.78(%)	49.37 (%)	49.71(%)	49.84 (%)	49.87(%)	49.93(%)	52.03 (%)
	Config 2	36.59 (%)	46.04 (%)	48.03 (%)	49.11 (%)	49.51(%)	49.74(%)	50.05 (%)
	Config 3	16.15 (%)	33.59 (%)	42.42 (%)	46.32 (%)	48.12 (%)	49.04(%)	49.29 (%)
	Config 4	0 (%)	10.81 (%)	28.38 (%)	39.93 (%)	44.82 (%)	47.5(%)	49.11 (%)
	Config 5	0 (%)	0 (%)	20.4 (%)	35.29 (%)	42.65 (%)	46.32 (%)	49.02 (%)
	Config 6	0 (%)	0 (%)	19.92 (%)	35.01 (%)	41.12 (%)	46.23 (%)	48.17 (%)
32 fog nodes	Config 1	48.78(%)	49.37 (%)	49.71(%)	49.84 (%)	49.87(%)	49.93(%)	52.03 (%)
	Config 2	37.01 (%)	46.08 (%)	48.95 (%)	49.47 (%)	49.73 (%)	49.86 (%)	51.02 (%)
	Config 3	16.72 (%)	33.11 (%)	44.15 (%)	47.05 (%)	48.36(%)	49.17(%)	49.53 (%)
	Config 4	0 (%)	4.96 (%)	31.55 (%)	40.7 (%)	45.35 (%)	47.65 (%)	49.27 (%)
	Config 5	0 (%)	0 (%)	21.1 (%)	37.2 (%)	43.44 (%)	46.82 (%)	49.19 (%)
	Config 6	0 (%)	0 (%)	19.81 (%)	35.11 (%)	42.42 (%)	45.91 (%)	48.02 (%)
64 fog nodes	Config 1	48.78(%)	49.37 (%)	49.71(%)	49.84 (%)	49.87(%)	49.93(%)	52.03 (%)
	Config 2	37.09 (%)	46.1 (%)	48.94 (%)	49.73 (%)	49.86 (%)	49.89 (%)	51.61 (%)
	Config 3	13.31 (%)	33.49 (%)	44.23 (%)	48.92 (%)	49.52 (%)	49.82(%)	49.41 (%)
	Config 4	0 (%)	3.37 (%)	31.44 (%)	44.89 (%)	47.41 (%)	48.42 (%)	49.36 (%)
	Config 5	0 (%)	0 (%)	0 (%)	42.69 (%)	46.31 (%)	47.67 (%)	49.31 (%)
	Config 6	0 (%)	0 (%)	0 (%)	42.32 (%)	45.22 (%)	47.21 (%)	49.23 (%)

7.7 Conclusion

The results in this chapter show that the use of a large number of fog nodes placed close to the data sources provides the parallelism needed to improve response times. The parallelism is through multiple fog nodes concurrently analyzing a subset of sensor data e.g., camera data, car data. The applications in Section 7.5 only require the analysis of data from a local area in order to make decisions. The data analysis can be done by a single fog node. However, other scenarios require data from an area where the use of one fog node would become a bottleneck. In this case it is beneficial to have a fog nodes cover a relatively small area representing a local region. These fog nodes send data to another fog node that does analysis of data sent by multiple fog nodes. This represents a hierarchical configuration of fog nodes. Section 7.6 showed the advantages of doing so but the results were mixed where the use of a hierarchical configuration of fog nodes did not always improve performance. The challenge is determining the placement of query operators among the fog nodes.

Chapter 8

Proposed Embedding Algorithms

The results in Section 7.5 show that the use of a large number of fog nodes placed close to the data sources provides the parallelism needed to improve response times especially for large amounts of data. The parallelism is through multiple fog nodes concurrently analyzing a subset of sensor data e.g., camera data, car data. The applications in Section 7.5 only require the analysis of data from a local area in order to make decisions. The data analysis can be done by a single fog node. However, other scenarios require data from an area where the use of one fog node would become a bottleneck. In this case it is beneficial to have a fog nodes cover a relatively small area representing a local region. These fog nodes send data to another fog node that does analysis of data sent by multiple fog nodes. This represents a hierarchical configuration of fog nodes. Section 7.6 showed the advantages of doing so but the results were mixed where the use of a hierarchical configuration of fog nodes did not always improve performance. The challenge is determining the placement of query operators among the fog nodes.

This chapter is organized as following: Section 8.1 discusses the proposed algorithms for analyzing query graphs. Section 8.2 presents the proposed mapping algorithms that is used to find the host fog nodes for query operators.

8.1 Query Graph Reshaping Algorithms

The placement of query operators needs to consider that too much processing on a fog node may result in the overloading of a fog node but too much distribution typically means more data transmission which increases network latency. One approach to reducing communication cost is to place query operators representing non-aggregation operators (e.g. filter operations) with query operators representing aggregation operators on the same fog node. This allows for the reduction of data transmission.

8.1.1 Reconfiguration Query Graph Algorithm

Algorithm 8.1 takes as input the original query graph and produces a new graph that combines non-aggregation operators and aggregation operators. In line 11 a new vertex w is created. The set $F[w]$ represents the set of query nodes in G (the input query graph) that will be associated

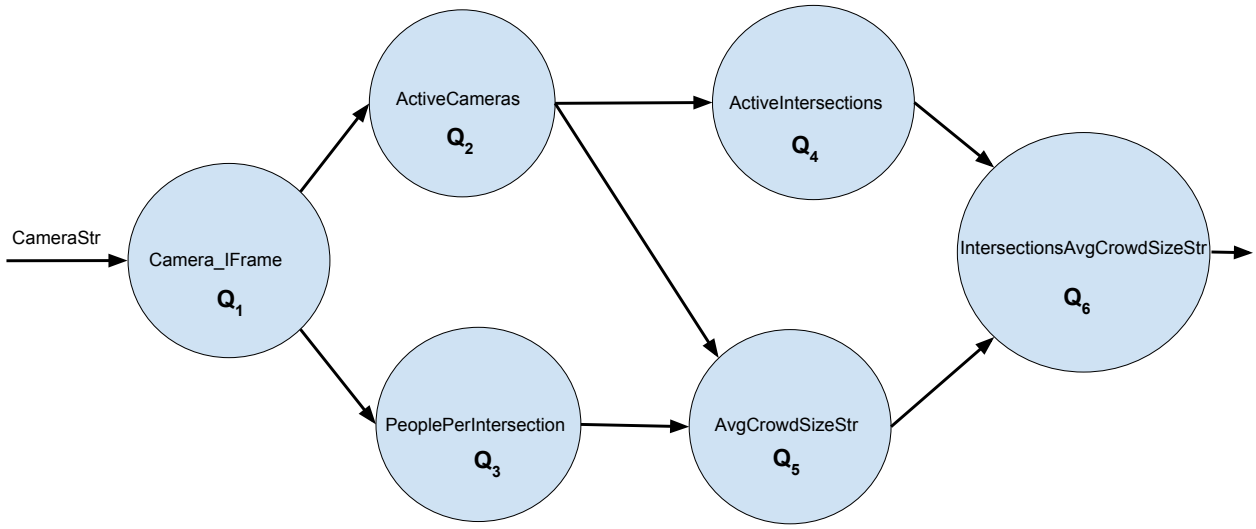

Figure 8.1: Camera Crows Size Scenario Query Graph

with vertex w. Lines 9 to 14 create the root nodes of G_{rec} and for each root node, w, a root node in G is assigned to $F[w]$. The input query graph is traversed in Lines 15-29 using BFS. For each edge (u, v), if v represents an aggregation operator then a new node,w, is created (line 20), v is assigned to $F[w]$ (line 21), w is added to V_{rec} (line 22) and an edge representing a link between the node in G_{rec} that has u and the node in G_{rec} that has v is added to E_{rec}.If v represents a non-aggregation operator then v will be associated with the same node as u.

Figure 8.1 represents an example input graph and Figure 8.2 represents an example output graph. More details of the query associated with Figure 8.1 can be found in Section 6.3.4 of chapter 6.

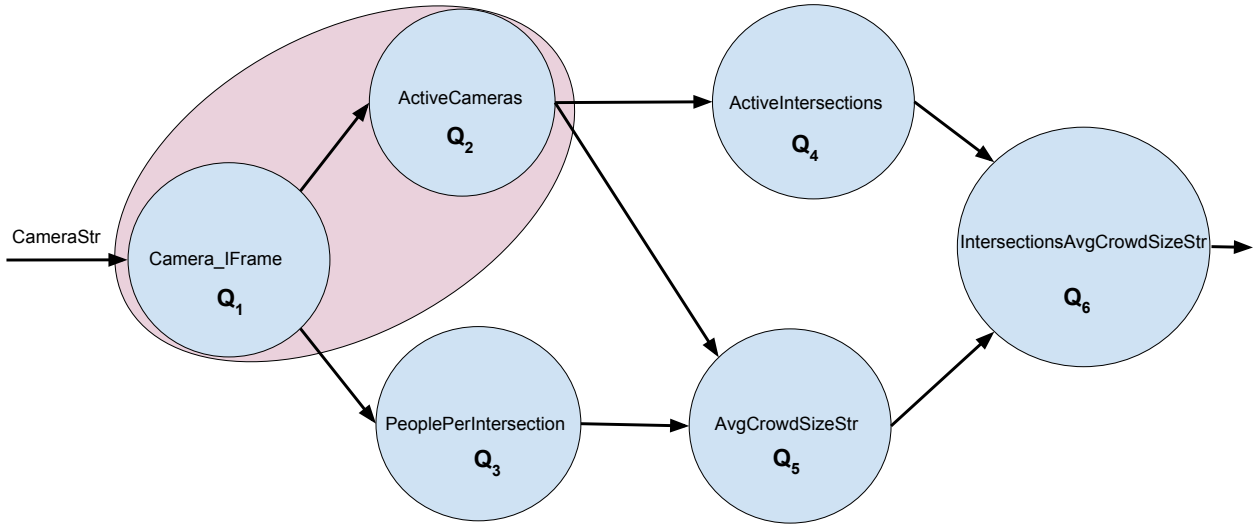

Figure 8.2: Reconfigured Query Graph

Algorithm 8.1 Query Graph Reconfiguration Algorithm

INPUT $G_Q = (V, E)$: A Query Graph
OUTPUT $G_{rec} = (V_{rec}, E_{rec})$: Reconfigured Query Graph
$F[x]$: Each element of F is a node of G_{rec} and represents the set of queries of G associated with the node.
$OP(x)$: The function returns the operators used in query x
$ROOT()$: Returns the root nodes of a given graph
queue : A queue for Breadth-First Search
$H[x]$: $H[x]$ is the node that x in G is associated with in the graph G_{rec}
$queue = NULL$
for all $u \in ROOT(G_Q)$ **do**
    $queue.Enqueue(u)$
    $w = createNode()$
    $F[w] = \{u\}$
    $V_{rec} = V_{rec} \cup w$
end for
while *queue* is not empty **do**
    $u = queue.DEQUEUE()$
    for all $(u, v) \in E$ **do**
        $queue.Enqueue(v)$
        if $OP(v) \in AggregateOperators$ **then**
            $w = createNode()$
            $F[w] = v$
            $V_{rec} = V_{rec} \cup \{w\}$
            $E_{rec} = E_{rec} \cup (H[u], w)$
        else if $OP(v) \in NonAggregateOperators$ **then**
            $H[u] = H[u] \cup \{v\}$
            $F[H[u]] = F[H[u]] \cup F[v]$
        end if
    end for
end while
Return G_{rec}

8.1.2 Adjustment of Reconfigured Query Graph Algorithm

In a reconfigured query graph (output of algorithm 8.1) there might be redundant data flow among query vertices. For example, consider query vertices Q_4, and Q_5 in Fig. 8.1. Q_4, and Q_5 are children of the query vertex Q_2 and receive the same input data flow from Q_2. This causes redundant flow of data among query vertices. Therefore algorithm 8.2 was proposed which applies an adjustment to a reconfigured query graph. In Fig. 8.2 vertex Q_2 sends the same data flow to Q_4 and Q_5. Accordingly algorithm 8.2 is proposed to cope with the aforementioned problem. Algorithm 8.2 merges children if any vertex of the graph produced in by Algorithm 8.1 of they receive the same data-flow.

The Query Graph Adjustment algorithm reduces the transfer of redundant data flow among query vertices. Algorithm 8.2 merges aggregate sibling-vertices (any two or more query vertices with a common parent are considered as sibling vertices) that receive the same data flow from the common parent. A vertex in a query graph can have no parent, one parent, or more than one parent. The Query Graph Adjustment algorithm uses the following steps to adjust the given reconfigured query graph.

1. Algorithm 8.2 starts from the root(s) of the reconfigured query graph.
2. At each level of the reconfigured query graph:
   (a) If a vertex has more than one child then algorithm 8.2 determines the children.
   (b) If there is no vertex with more than one child then algorithm 8.2 goes to the next level in the reconfigured query graph.
3. For those vertices with more than one child, the Query Graph Adjustment algorithm determines the type of input data flow to each child. If there are two or more children that receive the same type of data flow then the children are merged together.

In line 12 a new vertex w is created. The set $F[w]$ represents the set of query nodes that will be associated with vertex w. Lines 10 to 15 create the root nodes of G_{adj} and for each root node, w, a root node in G_{rec} is assigned to $F[w]$. The input reconfigured query graph is traversed in Lines 16-49 using BFS. In lines 18 to 23 if vertex u has more than one child then the children of u is added to the queue and V_{temp} to determine the existence of the data flow redundancy among children of vertex u. In a query graph an edge between two vertices represents a data flow. In lines 30 to 42 the algorithm identifies the siblings that receive the same data flow from their parent and merges them.

Figure 8.3 presents the output of of applying algorithm 8.2 to the graph shown in Figure 8.2.

Algorithm 8.2 Query Graph Adjustment Algorithm

INPUT $G = (V, E)$: Original Query Graph
INPUT $G_{rec} = (V_{rec}, E_{rec})$: A Reconfigured Query Graph
OUTPUT $G_{adj} = (V_{adj}, E_{adj})$: Adjustment Query Graph
$F_{adj}[x]$: Each element of F_{adj} is a node of G_{adj} and represents the set of queries of G_{rec} associated with the node.
$SubVer(x)$: The function returns the sub-vertices in vertex x
$ROOT()$: Returns the root nodes of a given graph
queue : A queue for Breadth-First Search
$H_{adj}[x]$: $H_{adj}[x]$ is the set of node(s) that x in G_{rec} is associated with in the graph G_{adj}
$queue = NULL$
for all $u \in ROOT(G_{rec})$ **do**
    $queue.Enqueue(u)$
    $w = createNode()$
    $F_{adj}[w] = \{u\}$
    $V_{adj} = V_{adj} \cup w$
end for
while *queue* is not empty **do**
    $u = queue.DEQUEUE()$
    if $\#Children(u) > 1$ **then**
        $V_{temp} = NULL$
        for all $(u, v) \in E_{rec}$ **do**
            $queue.ENQUEUE(v)$
            $V_{temp} = V_{temp} \cup v$
        end for
        $q = queue.DEQUEUE()$
        $V_{temp} = V_{temp} \cup (Children(q))$
        $w = createNode()$
        $F_{adj}[w] = \{q\}$
        $V_{adj} = V_{adj} \cup \{w\}$
        $E_{adj} = E_{adj} \cup (H[u], w)$
        for all $p \in V_{temp}$ **do**
            if $p \neq q$ **then**
                for all $x \in SubVer(u)$ **do**
                    if $(\exists y \in SubVer(q) \mid (x, y) \in E)$ && $(\exists z \in SubVer(p) \mid (x, z) \in E)$ **then**
                        $H_{adj}[w] = H_{adj}[w] \cup \{p\}$
                        $F_{adj}[H[w]] = F_{adj}[H_{adj}[w]] \cup F_{adj}[p]$
                        $V_{temp} = V_{temp}$ - p
                        $V_{temp} = V_{temp}$ - q
                        $queue.REMOVE(p)$
                    end if
                end for
            end if
        end for
    else if $\#Children(u) \leq 1$ **then**
        $w = createNode()$
        $F_{adj}[w] = \{q\}$
        $V_{adj} = V_{adj} \cup \{w\}$
        $E_{adj} = E_{adj} \cup (H_{adj}[u], w)$
    end if
end while
Return G_{adj}

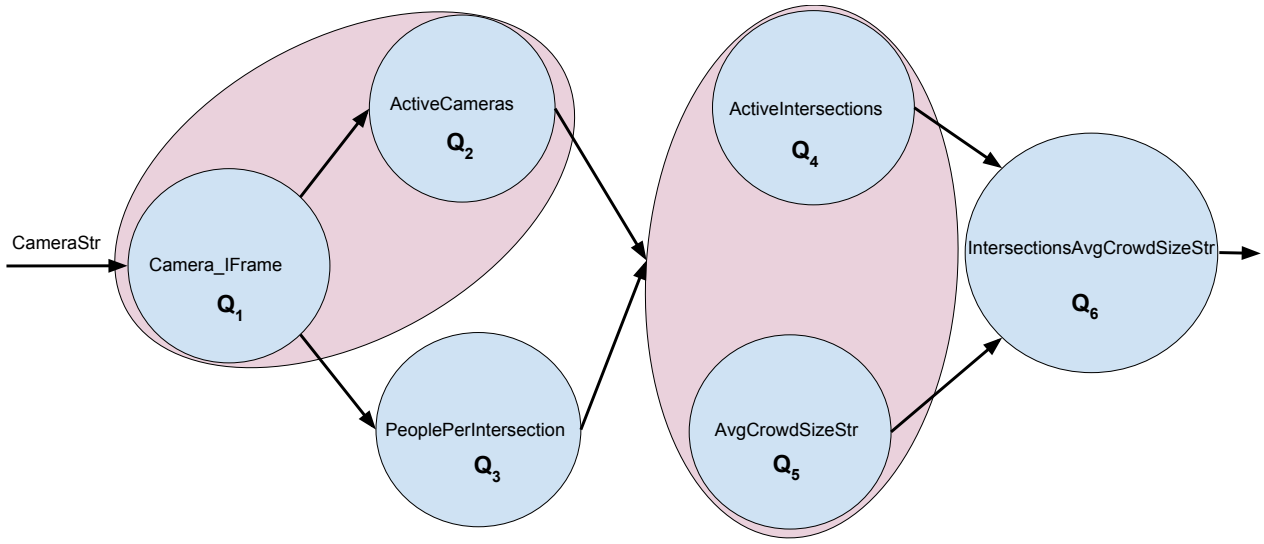

Figure 8.3: Adjusted Query Graph

8.2 Mapping Algorithm

This section describes the proposed mapping algorithms for mapping a given query graph into a set of fog nodes. The fog nodes are organized hierarchically. Fog nodes closer to the data sources have less computational power compare than those closer to the network core [194][128].

We categorise query vertices into anchor vertices and floating vertices. An anchor vertex is a vertex with no incoming edges or with no outgoing edges, and is typically assigned to a level 0 fog. A floating vertex is a vertex which is not an anchor vertex. Consider the query graph in Fig. 8.4. In Fig. 8.4 q'_1 and q'_4 are anchor vertices and q'_2, and q'_3 are floating vertices. Algorithm 8.3 focuses on mapping the floating vertices.

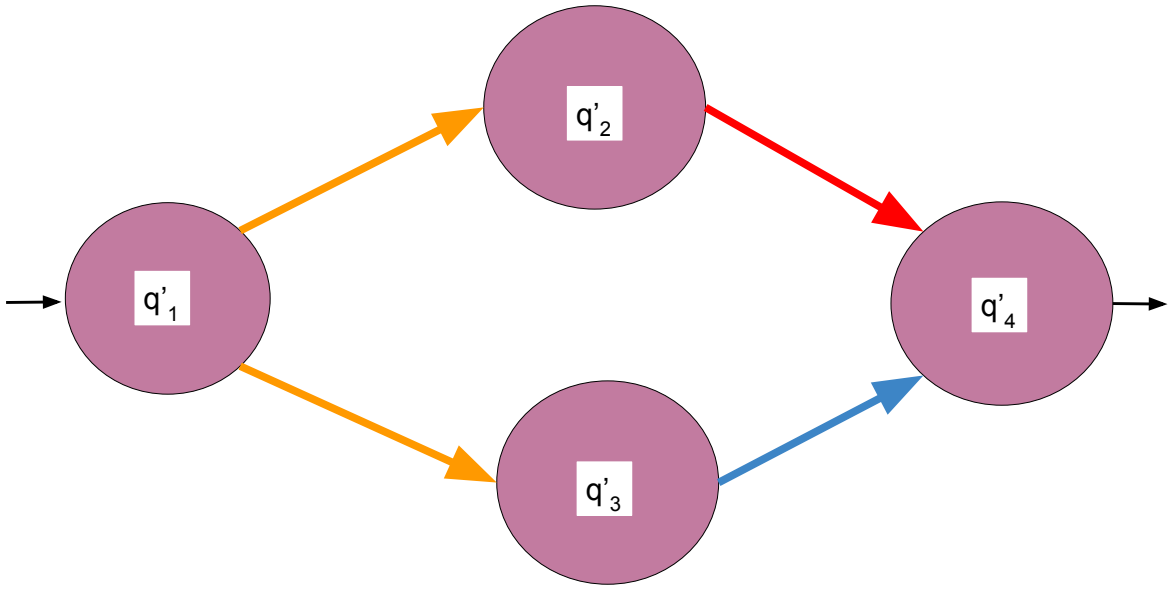

Figure 8.4: Example of a Query Graph

8.2.1 Proposed Mapping Algorithm

Algorithm 8.3 assumes that the anchor nodes have been assigned to a fog node. Anchor nodes with no incoming edges are assigned to level-0 fog nodes and anchor nodes with no outgoing edges are assigned to a fog node at the highest level. Lines 11 to 13 ensure that all anchor nodes are the first to be put into $queue_Q$ and thus enables the algorithm to assign vertices in V_Q starting from vertices that receive data from an anchor node. In lines 19 to 23 the children of u are added to $queue_Q$ so that they are considered in the next iteration of the WHILE loop. Lines 24 through 30 are used to find a fog node to host node v. Line 24 is used to determine the candidate fog nodes by considering all fog nodes that are not at the same level as the fog node hosting u. Lines 24-30 goes through each of the candidate fog nodes to determine a fog node that has the minimum cost for placing the vertex being considered for placement. It is worth noting that the input G_Q in algorithm 8.3 can be the original query graph or any of the reshaped query graphs (output of algorithms 8.2 or 8.2).

Algorithm 8.3 Embedding Algorithm

INPUT $G_Q = (V_Q, E_Q)$: A Query Graph
INPUT $G_T = (V_T, E_T)$: A Tree Overlay Network of Fog Nodes
INPUT $IM_V : IM_V = \{(x, y) : x \in V_Q, y \in P_c\}$
INPUT P_c: $P_c = \{x : x \in V_T$ and $foreign\_degIn(x) == TRUE\}$, $foreign\_degIn(y) = \{TRUE : y \in V_Q, x \notin V_Q, (x, y) \notin E_Q\}$ (Returns TRUE if $x \in V_Q$ receives raw data)
$Ancestor(x)$ returns all the ancestor vertices of $x \in V_T$ in G_T
$g(x, FM_V) = \{y : (x, y) \in FM_V\}$, $FM_V = \{(x, y) : x \in V_Q, y \in V_T\}$
$contains(x) = \{TRUE$: if a queue data structure has x, otherwise FALSE$\}$
OUTPUT FM_V
$FM_V = IM_V$
$queue_Q$ = NULL
for all $(x, y) \in IM_V$ **do**
    $queue_Q.Enqueue(x)$
end for
while $queue_Q ! = NULL$ **do**
    $u = queue_Q.Dequeue()$
    $candidateFogNodes = \emptyset$
    $minCostFog = NULL$
    $minCostValue = \infty$
    for all $(u, v) \in E_Q$ **do**
        if $!queue_Q.contains(v)$ **then**
            $queue_Q.Enqueue(v)$
        end if
    end for
    $candidateFogNodes = Ancestor(g(u, FM_V)) \cup g(u, FM_V)$
    for all $f \in candidateFogNodes$ **do**
        if $minCostValue > C_f((u, v))$ **then**
            $minCostValue = C_f((u, v))$
            $minCostFog = f$
        end if
    end for
    $FM_V = FM_V \cup (v, minCostFog)$
end while
Return FM_V

Chapter 9

Performance Evaluation

This chapter presents an evaluation of the algorithms proposed in Chapter 8. The metrics used are the same as those introduced in Section 7.1 of chapter 7.

This chapter is organized as follows: Section 9.1 describes the configuration of the simulation environment. Section 9.2 presents the results for the congested highway notification scenario. Section 9.3 presents the results for the camera surveillance scenario.

9.1 Configuring the Simulation Environment

This section describes the simulation environment configurations for the following: (1) organization of fog nodes, (2) three cost functions to be used by the mapping algorithm, and (3) applications.

9.1.1 Fog Node Networks Used

The fog nodes are organized as a tree where level 0 fog nodes directly receive data from the local data sources, level 1 fog nodes receive data from a subset of level 0 fog nodes, etc. Equation 9.1 is used to estimate the maximum height (number of levels) in a tree. In Equation 9.1, h represents the height of a tree with N nodes, and K represents the maximum number of children. With a fixed number of fog nodes as the maximum number of children increases the number height of the tree decreases. In the simulation, not all of the fog nodes have the maximum number of children allowed.

$$h = \lceil \log_k((k-1) \times N + 1) - 1 \rceil \tag{9.1}$$

- **Tree 1:** Tree 1 (total number of fog nodes is 256)
  - Maximum number of children for each fog node is 2
  - Number of levels is 8
- **Tree 2:** Tree 2 (total number of fog nodes is 256)
  - Maximum number of children for each fog node is 3

  - Number of levels is 6

- **Tree 3:** Tree 3 (total number of fog nodes is 256)
  - Maximum number of children for each fog node is 4
  - Number of levels is 5

Tree 1 is considered to be a tall tree and Tree 3 is considered to be a short tree. A shorter tree means that the non-leaf fog nodes communicate with more lower level fog nodes. Level 1 fog nodes communicate with more collector fog nodes which cover a broader geographical area. As a result the number of hops between fog nodes in two consecutive levels is higher in a shorter tree compared to a taller tree.

9.1.2 Applications

The applications described in Chapter 6 are used.

9.1.3 Cost Functions for Mapping Algorithm

Chapter 7 showed the possible benefits of using multiple levels in a fog-tree hierarchy. The cost functions presented in this section represent different approaches for mapping nodes in the query graph to nodes in the fog-tree hierarchy. These cost functions are used by the mapping algorithm presented in Chapter 8. The cost functions are based on Equation 6.3.

- C_1**:** With the cost function, C_1, the mapping algorithm maps all the nodes at the same level in the query graph to the same level in the fog-tree hierarchy. In addition, the cost function is designed such that for edge (q_i, q_j) of the query graph the length of the path between the nodes in the fog-tree hierarchy that q_i and q_j are mapped to is minimized as much as possible. The weights used in C_1 were selected such that for any (q_i, q_j) the cost of assigning q_i to a level k fog node and assigning q_j to a level k+1 fog node is less than assigning q_i to a level k fog node and assigning q_j to a level k+2 fog node. Furthermore any q_i at level 0 in the query graph is assigned to a fog node at level 0. The weight value used for the traffic congestion application is 0.16 and for the camera surveillance is 0.23.

- C_2**:** With the cost function, C_2, the mapping algorithm maps query vertices at different levels of the query graph to fog nodes at different levels in the fog-tree hierarchy i.e., for each edge (q_i, q_j) in the query graph q_i and q_j are mapped to different levels in the fog-tree hierarchy. High volume edges are mapped such that the number of hops are minimized as much as possible i.e., if (q_i, q_j) is a high volume edge and q_i is mapped to level k then the cost of mapping q_j to level k+1 is less than the cost of mapping q_i to level k+2. Compared to C_1, C_2 is designed to distribute the processing across more nodes such that a fog node is not overwhelmed with computation but also tries to minimize the transfer of data. The weight value used for the traffic congestion application is 0.28 and for the camera surveillance is 0.37.

- C_3**:** With the cost function, C_3, the mapping algorithm maps each query vertex in a level of a query graph with one level of a tree that provides the lowest response time. C_3 considers the importance of value for execution time and network end-to-end delay equally. The weight value used for the traffic congestion application is 0.28 and for the camera surveillance is 0.5.

9.2 Congested Highway Notification Scenario Results

In this section, we show the evaluation results for the Local Congested Highway Notification scenario. The information about details of query operators that are used in query graph for the congested highway notification scenario can be found in Appendix B.

Table 9.1 shows the embedding distortion for C_1, C_2, and C_3 for embedding the query operators that used by the congested highway notification scenario. The definition for embedding distortion can be found in Chapter 6. Distortion values that are closer to 1 means that query operators are placed closer to the data sources than mappings that result in lower distortion values.

Table 9.1: Maximum Embedding Distortion for Congested Highway Notification Scenario

	Tree 1			Tree 2			Tree 3		
	Original QG	Reconfigured QG	Adjusted QG	Original QG	Reconfigured QG	Adjusted QG	Original QG	Reconfigured QG	Adjusted QG
C_1	0.17	0.22	0.22	0.13	0.17	0.17	0.095	0.11	0.11
C_2	0.11	0.13	0.17	0.11	0.11	0.13	0.095	0.095	0.095
C_3	0.095	0.11	0.11	0.11	0.11	0.11	0.095	0.095	0.095

9.2.1 Average Execution Time

Tables 9.2 and 9.3 show the average execution time used to process the original query graph, reconfigured query graph, and adjusted query graph for the cost functions. In this thesis collector fog nodes (CFNs) and level-0 fog nodes are used interchangeably.

Effectiveness of Collector Fog Nodes: The results in Table 9.2 show that when the number of CFNs is 50 or less the average execution time decreases by increasing the CFNs. When the number of CFNs is 60 or higher the average execution time starts to increase since the number of fog notes associated with a level 1 fog node increases. A similar pattern is seen in Table 9.3 where the number of vehicles is 2000.

Effectiveness of Fog-Tree Hierarchy: Tree 3, representing the shortest tree, has the lowest execution time as the result of providing more fog nodes closer to data sources than the other two trees.

Effectiveness of Algorithms 8.1 and 8.2: From Table 9.2 and 9.3 we see that the original query graph results in less execution time compared to the reconfigured and the adjusted query graph since in the reconfigured or adjusted query graph some of query vertices are merged and as a result more processing power is required to process the merged vertices.

Comparison of Cost Functions: From Table 9.2 and 9.3 we see that cost function C_3 has lower execution time compared to C_1 and C_2. The cost function C_3 favours mapping query

Table 9.2: Average Execution Time-1000 Vehicles

		Tree 1			Tree 2			Tree 3		
	$\#CFNs$	Original QG	Reconfigured QG	Adjusted QG	Original QG	Reconfigured QG	Adjusted QG	Original QG	Reconfigured QG	Adjusted QG
C_1	2	18.915 (s)	33.809 (s)	33.889 (s)	17.926 (s)	31.834 (s)	31.587 (s)	15.24 (s)	27.945 (s)	27.873 (s)
	10	6.175 (s)	8.234 (s)	8.233 (s)	5.139 (s)	8.665 (s)	8.001 (s)	5.044 (s)	6.485 (s)	6.433 (s)
	20	6.763 (s)	7.786 (s)	7.726 (s)	5.653 (s)	6.829 (s)	5.829 (s)	4.096 (s)	4.114 (s)	4.987 (s)
	30	6.003 (s)	7.434 (s)	7.834 (s)	5.341 (s)	6.462 (s)	6.507 (s)	3.897 (s)	3.906 (s)	4.772 (s)
	40	5.871 (s)	8.821 (s)	8.834 (s)	5.302 (s)	6.175 (s)	6.831 (s)	3.681 (s)	3.682 (s)	4.802 (s)
	50	5.452 (s)	10.982 (s)	10.885 (s)	5.272 (s)	5.657 (s)	7.125 (s)	6.237 (s)	6.158 (s)	6.402 (s)
	60	13.775 (s)	13.428 (s)	13.728 (s)	11.393 (s)	9.694 (s)	9.704 (s)	8.247 (s)	9.102 (s)	9.822 (s)
	70	14.183 (s)	13.839 (s)	14.113 (s)	11.738 (s)	10.231 (s)	10.409 (s)	8.492 (s)	9.773 (s)	10.191 (s)
	80	17.084 (s)	16.352 (s)	16.412 (s)	14.175 (s)	13.853 (s)	13.923 (s)	11.033 (s)	11.539 (s)	11.621 (s)
C_2	2	17.401 (s)	31.104 (s)	31.177 (s)	16.491(s)	29.287(s)	29.064 (s)	14.028 (s)	25.704 (s)	25.643(s)
	10	5.681 (s)	7.575(s)	7.574(s)	4.727(s)	7.971(s)	7.36(s)	4.648(s)	5.966(s)	5.918(s)
	20	6.221 (s)	7.163 (s)	7.107(s)	5.206 (s)	6.282(s)	5.362(s)	3.768 (s)	3.784 (s)	4.584(s)
	30	5.522 (s)	6.839 (s)	7.207 (s)	4.913(s)	5.945 (s)	5.986(s)	3.585(s)	3.593 (s)	4.39 (s)
	40	5.401(s)	8.1153 (s)	8.127(s)	4.877 (s)	5.681 (s)	6.284(s)	3.386(s)	3.387 (s)	4.417(s)
	50	5.0158 (s)	10.103 (s)	10.014(s)	4.85(s)	5.204 (s)	6.555(s)	5.738(s)	5.665 (s)	5.889(s)
	60	12.673 (s)	12.353 (s)	12.629 (s)	10.481 (s)	8.918(s)	8.927(s)	7.587(s)	8.373 (s)	9.036(s)
	70	13.048 (s)	12.731(s)	12.983(s)	10.798(s)	9.412 (s)	9.576(s)	7.812(s)	8.991 (s)	9.375(s)
	80	15.717(s)	15.043 (s)	14.731 (s)	13.041(s)	12.836 (s)	12.809 (s)	10.15(s)	10.891 (s)	10.991(s)
C_3	2	16.645 (s)	29.751 (s)	29.822 (s)	15.774 (s)	28.013 (s)	27.796(s)	13.411 (s)	24.591 (s)	24.528 (s)
	10	5.434(s)	7.245 (s)	7.245(s)	4.522(s)	7.622 (s)	7.048 (s)	4.438(s)	5.708 (s)	5.664 (s)
	20	5.951 (s)	6.851 (s)	6.798 (s)	4.974 (s)	6.002 (s)	5.122(s)	3.608 (s)	3.622(s)	4.388 (s)
	30	5.282 (s)	6.541 (s)	6.893 (s)	4.708 (s)	5.656 (s)	5.726(s)	3.426 (s)	3.438(s)	4.199(s)
	40	5.166 (s)	7.762 (s)	7.773(s)	4.665 (s)	5.434 (s)	6.018(s)	3.239(s)	3.246(s)	4.226(s)
	50	4.797 (s)	9.664(s)	9.578 (s)	4.639 (s)	4.9786 (s)	6.27(s)	5.488(s)	5.414(s)	5.633(s)
	60	12.122 (s)	11.816 (s)	12.08 (s)	10.024 (s)	8.532 (s)	8.532 (s)	7.256 (s)	8.006(s)	8.646(s)
	70	12.484 (s)	12.178 (s)	12.419(s)	10.329 (s)	9.003 (s)	9.1592(s)	7.476 (s)	8.602 (s)	8.968(s)
	80	15.032 (s)	14.389 (s)	14.09 (s)	12.474 (s)	12.278 (s)	12.252 (s)	9.709 (s)	10.418 (s)	10.226 (s)

Table 9.3: Average Execution Time-2000 Vehicles

		Tree 1			Tree 2			Tree 3		
	$\#CFNs$	Original QG	Reconfigured QG	Adjusted QG	Original QG	Reconfigured QG	Adjusted QG	Original QG	Reconfigured QG	Adjusted QG
C_1	2	36.015 (s)	69.134 (s)	69.238 (s)	35.116 (s)	68.235 (s)	67.192 (s)	30.851 (s)	58.763 (s)	58.718 (s)
	10	10.146 (s)	15.182 (s)	15.478 (s)	9.004 (s)	14.044 (s)	14.031 (s)	8.164 (s)	12.175 (s)	12.163 (s)
	20	8.233 (s)	9.235 (s)	9.603 (s)	7.056 (s)	6.452 (s)	6.834 (s)	5.975 (s)	6.958 (s)	6.652 (s)
	30	7.099 (s)	8.137 (s)	8.525 (s)	6.047 (s)	5.396 (s)	5.756 (s)	5.008 (s)	5.981 (s)	5.661 (s)
	40	7.959 (s)	9.047 (s)	9.445 (s)	6.369 (s)	5.827 (s)	6.185 (s)	4.987 (s)	5.953 (s)	5.63 (s)
	50	12.812 (s)	11.613 (s)	11.682 (s)	10.648 (s)	10.734 (s)	10.013 (s)	8.857 (s)	8.523 (s)	8.835 (s)
	60	13.951 (s)	14.034 (s)	14.905 (s)	11.994 (s)	10.451 (s)	10.804 (s)	10.008 (s)	9.102 (s)	9.048 (s)
	70	14.97 (s)	15.1 (s)	15.993 (s)	13.082 (s)	11.497 (s)	11.822 (s)	10.889 (s)	9.864 (s)	9.72 (s)
	80	17.971 (s)	17.849 (s)	17.902 (s)	14.245 (s)	15.324 (s)	15.157 (s)	12.033 (s)	12.089 (s)	12.284 (s)
C_2	2	32.415 (s)	62.206(s)	62.312 (s)	31.604 (s)	61.415(s)	60.472(s)	27.769(s)	52.867(s)	52.842(s)
	10	9.114 (s)	13.668(s)	13.932 (s)	8.106(s)	12.636(s)	12.629(s)	7.346(s)	10.955(s)	10.947 (s)
	20	7.407(s)	8.315(s)	8.642(s)	6.304(s)	5.806 (s)	6.106(s)	5.375(s)	6.222 (s)	5.986 (s)
	30	6.381(s)	7.323(s)	7.675(s)	5.443(s)	4.84(s)	5.184(s)	4.507(s)	5.389(s)	5.094 (s)
	40	7.161(s)	8.143(s)	8.505(s)	5.732(s)	5.243(s)	5.565(s)	4.483(s)	5.357(s)	5.06 (s)
	50	11.508 (s)	10.417(s)	10.518 (s)	9.52(s)	9.666(s)	9.011(s)	7.913 (s)	7.677(s)	7.955 (s)
	60	12.559 (s)	12.606 (s)	13.415(s)	10.796(s)	9.409(s)	9.726(s)	9.072(s)	8.118(s)	8.142 (s)
	70	13.473(s)	13.59(s)	14.39(s)	11.77 (s)	10.347(s)	10.639 (s)	9.801(s)	8.877(s)	8.748(s)
	80	16.139(s)	16.041(s)	16.118 (s)	12.805 (s)	13.796(s)	13.641 (s)	10.897(s)	10.801 (s)	11.056(s)
C_3	2	30.979 (s)	59.454(s)	59.548(s)	30.196(s)	58.681(s)	57.782 (s)	26.531(s)	50.538 (s)	50.448 (s)
	10	8.726 (s)	13.052 (s)	13.311(s)	7.744 (s)	12.074 (s)	12.066 (s)	7.004(s)	10.475 (s)	10.468 (s)
	20	7.088 (s)	7.941(s)	8.258 (s)	6.066 (s)	5.542(s)	5.824(s)	5.135 (s)	5.983(s)	5.722 (s)
	30	6.104 (s)	6.992 (s)	7.335 (s)	5.202(s)	4.64(s)	4.916 (s)	4.368(s)	5.146(s)	4.846(s)
	40	6.844(s)	7.782 (s)	8.122 (s)	5.474(s)	5.011 (s)	5.311(s)	4.282(s)	5.118(s)	4.848(s)
	50	11.012 (s)	9.988 (s)	10.046 (s)	9.158 (s)	9.224 (s)	8.618 (s)	7.602 (s)	7.378 (s)	7.581 (s)
	60	11.996 (s)	12.069(s)	12.83 (s)	10.314(s)	8.986(s)	9.294 (s)	8.608(s)	7.872(s)	7.788(s)
	70	12.874(s)	12.98 (s)	13.758 (s)	11.252 (s)	9.882(s)	10.192(s)	9.354(s)	8.484(s)	8.352(s)
	80	15.455 (s)	15.354(s)	15.392 (s)	12.257 (s)	13.174(s)	13.235 (s)	10.348(s)	10.594 (s)	10.764(s)

vertices of a given query graph to upper levels of the fog-node hierarchy. These fog nodes have more processing power compared to fog nodes in the lower levels of the fog node hierarchy. The results also show that since C_2 uses more levels than C_1 that more powerful computing power is available resulting in lower execution times.

9.2.2 Average End-to-End Delay

Tables 9.4 and 9.5 show that the average end-to-end delay for processing of the original query graph, reconfigured query graph, and adjusted query graph.

Effectiveness of Collector Fog Nodes: The results in Table 9.4 show that the end-to-end delay decreases as the number of CFNs increases until the number of CFNs reaches 60. At that point the average end-to-end delay increases as the number of CFNs increases. This trend is true for each of the trees. A similar pattern is seen in Table 9.5 where the number of vehicles is 2000

Table 9.4: Average End-to-End Delay-1000 Vehicles

		Tree 1			Tree 2			Tree 3		
	$\#CFNs$	Original QG	Reconfigured QG	Adjusted QG	Original QG	Reconfigured QG	Adjusted QG	Original QG	Reconfigured QG	Adjusted QG
C_1	2	0.036862 (s)	0.036745 (s)	0.03678 (s)	0.183858 (s)	0.165876 (s)	0.129411 (s)	0.227913 (s)	0.172311 (s)	0.33047 (s)
	10	0.01192 (s)	0.01028 (s)	0.01003 (s)	0.012892 (s)	0.03883 (s)	0.03065 (s)	0.017645 (s)	0.03923 (s)	0.03585 (s)
	20	0.008793 (s)	0.00552 (s)	0.00443 (s)	0.023532 (s)	0.023706 (s)	0.018542 (s)	0.042153 (s)	0.02823 (s)	0.02123 (s)
	30	0.008102 (s)	0.004913 (s)	0.003971 (s)	0.028924 (s)	0.022819 (s)	0.018301 (s)	0.047153 (s)	0.029931 (s)	0.022501 (s)
	40	0.007589 (s)	0.004863 (s)	0.003809 (s)	0.030951 (s)	0.022706 (s)	0.018103 (s)	0.051003 (s)	0.032197 (s)	0.025538 (s)
	50	0.006757 (s)	0.00487 (s)	0.00383 (s)	0.051255 (s)	0.022158 (s)	0.020448 (s)	0.056923 (s)	0.03632 (s)	0.03189 (s)
	60	0.007512 (s)	0.004812 (s)	0.003662 (s)	0.052203 (s)	0.02383 (s)	0.01262 (s)	0.06475 (s)	0.040564 (s)	0.03468 (s)
	70	0.011093 (s)	0.005102 (s)	0.003959 (s)	0.073181 (s)	0.063201 (s)	0.018355 (s)	0.110301 (s)	0.07301 (s)	0.041305 (s)
	80	0.011264 (s)	0.008264 (s)	0.007382 (s)	0.080544 (s)	0.051048 (s)	0.022089 (s)	0.11282 (s)	0.079144 (s)	0.07104 (s)
C_2	2	0.0409168 (s)	0.0407865 (s)	0.040825(s)	0.204073(s)	0.184038(s)	0.143634(s)	0.25197 (s)	0.191265(s)	0.36682(s)
	10	0.023231(s)	0.011418 (s)	0.011133(s)	0.014208(s)	0.0431013(s)	0.0340215(s)	0.019536(s)	0.043545(s)	0.039793(s)
	20	0.009763 (s)	0.006122(s)	0.004917(s)	0.026085 (s)	0.026307(s)	0.020535(s)	0.046731 (s)	0.031335(s)	0.023565(s)
	30	0.008993(s)	0.005453 (s)	0.004481(s)	0.032079(s)	0.025329 (s)	0.020313(s)	0.052281 (s)	0.033221(s)	0.024976(s)
	40	0.008427(s)	0.005393 (s)	0.004227 (s)	0.034356 (s)	0.025197 (s)	0.020091(s)	0.05661(s)	0.035738(s)	0.028347(s)
	50	0.0075027 (s)	0.005457 (s)	0.00425(s)	0.056832 (s)	0.024531 (s)	0.022644 (s)	0.063159 (s)	0.040315(s)	0.035397(s)
	60	0.008338 (s)	0.005341 (s)	0.004062(s)	0.057942 (s)	0.026418(s)	0.01332(s)	0.07104(s)	0.044955 (s)	0.038494 (s)
	70	0.012313(s)	0.005663(s)	0.004399(s)	0.081141(s)	0.070152 (s)	0.020313 (s)	0.122433 (s)	0.08103(s)	0.045848(s)
	80	0.012503(s)	0.009173 (s)	0.008192 (s)	0.089355 (s)	0.056654 (s)	0.02442(s)	0.12432 (s)	0.087801(s)	0.078854(s)
C_3	2	0.0423913 (s)	0.042256 (s)	0.042297 (s)	0.2114367 (s)	0.1907574(s)	0.148826 (s)	0.26209 (s)	0.198145 (s)	0.3795 (s)
	10	0.023708 (s)	0.011822 (s)	0.01265 (s)	0.014825(s)	0.044654 (s)	0.035247 (s)	0.0203742 (s)	0.04511(s)	0.04025 (s)
	20	0.010111(s)	0.00632(s)	0.00506 (s)	0.0270618 (s)	0.027255 (s)	0.021323 (s)	0.04859(s)	0.032465(s)	0.02415 (s)
	30	0.009317 (s)	0.00564(s)	0.004566 (s)	0.03326(s)	0.026241(s)	0.021045(s)	0.05422 (s)	0.03406(s)	0.025875 (s)
	40	0.008723 (s)	0.00552 (s)	0.00437 (s)	0.035595 (s)	0.026105(s)	0.020815 (s)	0.05865 (s)	0.037026(s)	0.029325 (s)
	50	0.007775 (s)	0.00552(s)	0.00437(s)	0.058945 (s)	0.025481(s)	0.023512(s)	0.065445(s)	0.041768 (s)	0.03565 (s)
	60	0.00868 (s)	0.005538 (s)	0.00345 (s)	0.06003 (s)	0.027405(s)	0.014513 (s)	0.074462 (s)	0.046648 (s)	0.03988 (s)
	70	0.012756 (s)	0.005863 (s)	0.004555 (s)	0.084151 (s)	0.07268 (s)	0.021045 (s)	0.126845 (s)	0.08395(s)	0.047505 (s)
	80	0.01293(s)	0.009503 (s)	0.008483 (s)	0.092626 (s)	0.058702(s)	0.0253 (s)	0.129743 (s)	0.091016 (s)	0.08165 (s)

Table 9.5: Average End-to-End Delay-2000 Vehicles

		Tree 1			Tree 2			Tree 3		
	$\#CFNs$	Original QG	Reconfigured QG	Adjusted QG	Original QG	Reconfigured QG	Adjusted QG	Original QG	Reconfigured QG	Adjusted QG
C_1	2	0.073465 (s)	0.07233 (s)	0.071232 (s)	0.347518 (s)	0.330676 (s)	0.25829 (s)	0.454846 (s)	0.344011 (s)	0.35481 (s)
	10	0.02536 (s)	0.02143 (s)	0.02031 (s)	0.02556 (s)	0.07179 (s)	0.056428 (s)	0.122048 (s)	0.073578 (s)	0.07071 (s)
	20	0.01154 (s)	0.00954 (s)	0.00837 (s)	0.027165 (s)	0.040872 (s)	0.032296 (s)	0.06485 (s)	0.044721 (s)	0.04265 (s)
	30	0.00988 (s)	0.007731 (s)	0.007131 (s)	0.025502 (s)	0.033197 (s)	0.029701 (s)	0.066091 (s)	0.044531 (s)	0.041771 (s)
	40	0.009731 (s)	0.006891 (s)	0.006935 (s)	0.041475 (s)	0.028977 (s)	0.028302 (s)	0.066103 (s)	0.044392 (s)	0.040013 (s)
	50	0.009353 (s)	0.00625 (s)	0.005143 (s)	0.059226 (s)	0.02923 (s)	0.025243 (s)	0.066274 (s)	0.042648 (s)	0.03768 (s)
	60	0.010203 (s)	0.008763 (s)	0.007609 (s)	0.060752 (s)	0.04228 (s)	0.036724 (s)	0.070331 (s)	0.058107 (s)	0.05267 (s)
	70	0.011873 (s)	0.008898 (s)	0.008572 (s)	0.072133 (s)	0.048703 (s)	0.039981 (s)	0.120985 (s)	0.097827 (s)	0.058313 (s)
	80	0.023812 (s)	0.023035 (s)	0.020151 (s)	0.088492 (s)	0.054969 (s)	0.043854 (s)	0.23804 (s)	0.18809 (s)	0.078439 (s)
C_2	2	0.0896273(s)	0.088242(s)	0.086904 (s)	0.423976(s)	0.403424 (s)	0.3151138 (s)	0.55491212 (s)	0.419642(s)	0.4328682 (s)
	10	0.0309392 (s)	0.0261446(s)	0.024782 (s)	0.0311832 (s)	0.087588(s)	0.068846(s)	0.148886 (s)	0.089766 (s)	0.086262(s)
	20	0.0140788 (s)	0.011688(s)	0.0102114 (s)	0.0331413 (s)	0.04984(s)	0.039402(s)	0.079117(s)	0.054552 (s)	0.052033 (s)
	30	0.0120536 (s)	0.009418 (s)	0.008699 (s)	0.031114(s)	0.040504(s)	0.036235(s)	0.0806102(s)	0.054382(s)	0.0509602(s)
	40	0.011872(s)	0.008402(s)	0.0084607 (s)	0.050595 (s)	0.035351 (s)	0.034528 (s)	0.080646(s)	0.0541824 (s)	0.048818(s)
	50	0.011406 (s)	0.007625 (s)	0.006274 (s)	0.072272 (s)	0.035606 (s)	0.030796(s)	0.080854(s)	0.052036 (s)	0.0459696 (s)
	60	0.012447(s)	0.010608(s)	0.009289 (s)	0.0741144 (s)	0.051586(s)	0.044803(s)	0.085802(s)	0.070954 (s)	0.06254(s)
	70	0.014486(s)	0.010855 (s)	0.010454 (s)	0.088002(s)	0.059417(s)	0.048776 (s)	0.1476017(s)	0.1193484 (s)	0.0711486 (s)
	80	0.029064 (s)	0.0281027(s)	0.024542 (s)	0.107964 (s)	0.067062 (s)	0.053501 (s)	0.290408(s)	0.229498 (s)	0.09595(s)
C_3	2	0.093305(s)	0.091859 (s)	0.0904646 (s)	0.441347 (s)	0.419952(s)	0.328028 (s)	0.577654(s)	0.436893 (s)	0.450607 (s)
	10	0.03222(s)	0.027216(s)	0.025793(s)	0.032461 (s)	0.091173(s)	0.071663(s)	0.1550096(s)	0.093406(s)	0.089807(s)
	20	0.01465(s)	0.012115(s)	0.010629 (s)	0.034499 (s)	0.051904(s)	0.041012 (s)	0.0823595 (s)	0.056767(s)	0.054165(s)
	30	0.012547(s)	0.009818(s)	0.009056(s)	0.032387(s)	0.046019(s)	0.037727(s)	0.083937 (s)	0.05657(s)	0.053049(s)
	40	0.012358 (s)	0.008757 (s)	0.008807(s)	0.052673 (s)	0.036809(s)	0.0359434 (s)	0.083908(s)	0.056374 (s)	0.05081(s)
	50	0.011878 (s)	0.007937 (s)	0.006531(s)	0.075217 (s)	0.0371221(s)	0.0320861 (s)	0.084168(s)	0.054162(s)	0.047853 (s)
	60	0.012957(s)	0.011121 (s)	0.009663 (s)	0.077504 (s)	0.053656(s)	0.0466348 (s)	0.089337(s)	0.073799(s)	0.066899 (s)
	70	0.015078(s)	0.011304(s)	0.010884 (s)	0.091609 (s)	0.061858(s)	0.0507758(s)	0.153655(s)	0.124249 (s)	0.074057(s)
	80	0.030241 (s)	0.029254(s)	0.025591(s)	0.1123884 (s)	0.069806(s)	0.0556948 (s)	0.3023108 (s)	0.23883(s)	0.099617 (s)

although the effectiveness of CFNs decreases when the number of CFNs goes from 50 to 60.

Effectiveness of the Fog-Tree Hierarchy: Tables 9.4 and 9.5 show that Tree 1 has the lowest end-to-end delay compared to Tree 2 and Tree 3. Tree 1 is considered a tall tree and Tree 3 is considered a short tree. In Table 9.5 as the number of vehicles were doubled, the end-to-end delay did not significantly increase in the presence of a large number of collector fog nodes. In the other words, we can conclude that the by using hierarchy network of fog nodes we can keep end-to-end delay low when the number of data sources increases.

Effectiveness of Algorithm 8.1 and 8.2: From Tables 9.4 and 9.5 we see that the use of the original query graph results in the highest end-to-end delay compared to the use of the reconfigured and adjusted query graphs.

Comparison of Cost Functions C_1, C_2, and C_3: From Tables 9.4 and 9.5 we see that cost function C_1 has lower delay time compared to cost functions C_2 and C_3 . This is expected since the cost function C_1 favours the mapping of the query vertices as close as possible to the data

sources.

9.2.3 Average Response Time

Tables 9.6 and 9.7 shows the average response time for processing of the original query graph, reconfigured query graph, and adjusted query graph as the number of fog nodes and vehicles vary.

Table 9.6: Average Response Time-1000 Vehicles

		Tree 1			Tree 2			Tree 3		
	$\#CFNs$	Original QG	Reconfigured QG	Adjusted QG	Original QG	Reconfigured QG	Adjusted QG	Original QG	Reconfigured QG	Adjusted QG
C_1	2	18.951862 (s)	33.845745 (s)	33.92578 (s)	18.109858 (s)	31.999876 (s)	31.716411 (s)	15.467913 (s)	28.117311 (s)	28.20347 (s)
	10	6.176192 (s)	8.24428 (s)	8.24303 (s)	5.151892 (s)	8.70383 (s)	8.03165 (s)	5.061645 (s)	6.52423 (s)	6.46885 (s)
	20	6.771793 (s)	7.79152 (s)	7.73043 (s)	5.676532 (s)	6.852706 (s)	5.847542 (s)	4.138153 (s)	4.14223 (s)	5.00823 (s)
	30	6.011102 (s)	7.438913 (s)	7.837971 (s)	5.369924 (s)	6.484819 (s)	6.525301 (s)	3.944153 (s)	3.935931 (s)	4.794501 (s)
	40	5.878589 (s)	8.825863 (s)	8.837809 (s)	5.332951 (s)	6.197706 (s)	6.849103 (s)	3.732003 (s)	3.714197 (s)	4.827538 (s)
	50	5.458757 (s)	10.98687 (s)	10.88883 (s)	5.323255 (s)	5.679158 (s)	7.145448 (s)	6.293923 (s)	6.19432 (s)	6.43389 (s)
	60	13.782512 (s)	13.432812 (s)	13.731662 (s)	11.445203 (s)	9.71783 (s)	9.71662 (s)	8.31175 (s)	9.142564 (s)	9.85668 (s)
	70	14.194093 (s)	13.844102 (s)	14.116959 (s)	11.811181 (s)	10.294201 (s)	10.427355 (s)	8.602301 (s)	9.84601 (s)	10.232305 (s)
	80	17.095264 (s)	16.360264 (s)	16.419382 (s)	14.255544 (s)	13.904048 (s)	13.945089 (s)	11.14582 (s)	11.618144 (s)	11.69204 (s)
C_2	2	17.4427 (s)	31.14506(s)	31.2187(s)	16.6959(s)	29.4713(s)	29.2036(s)	14.272(s)	25.9006(s)	26.0099(s)
	10	5.6823 (s)	7.58669(s)	7.58549(s)	4.742(s)	8.0149(s)	7.39494(s)	4.66(s)	6.0097(s)	5.95815(s)
	20	6.23172 (s)	7.16924 (s)	7.11283(s)	5.2268(s)	6.3089(s)	5.3832(s)	3.81505 (s)	3.81621 (s)	4.6116(s)
	30	5.53175 (s)	6.84473 (s)	7.21168781 (s)	4.94579(s)	5.97036 (s)	6.0067(s)	3.6375(s)	3.6267 (s)	4.41521(s)
	40	5.40974(s)	8.12071(s)	8.131507(s)	4.91219 (s)	5.7061 (s)	6.30461(s)	3.44313(s)	3.42317 (s)	4.44618(s)
	50	5.02334(s)	10.10884 (s)	10.01845(s)	4.907 (s)	5.22897 (s)	6.57764(s)	5.8011 (s)	5.70567 (s)	5.9252 (s)
	60	12.6813 (s)	12.3591 (s)	12.63382 (s)	10.5395 (s)	8.94489(s)	8.94(s)	7.658(s)	8.418 (s)	9.07473(s)
	70	13.0606(s)	12.73754(s)	12.98835(s)	10.8801(s)	9.4826 (s)	9.5965(s)	7.935(s)	9.0721 (s)	9.4215(s)
	80	15.7297(s)	15.053(s)	14.73923 (s)	13.1303(s)	12.89341(s)	12.833(s)	10.274(s)	10.979(s)	11.0701(s)
C_3	2	16.6875 (s)	29.79417 (s)	29.864(s)	15.9863(s)	28.20467(s)	27.945382(s)	13.67329 (s)	24.7897 (s)	24.907 (s)
	10	5.43537 (s)	7.2577 (s)	7.257(s)	4.53714(s)	7.66985 (s)	7.07612 (s)	4.45909 (s)	5.7519 (s)	5.7012(s)
	20	5.96155(s)	6.858(s)	6.8039 (s)	5.0017(s)	6.0367 (s)	5.15084(s)	3.65307 (s)	3.6527(s)	4.41271 (s)
	30	5.29195(s)	6.5475 (s)	6.898(s)	4.73334 (s)	5.712801 (s)	5.7472(s)	3.48358 (s)	3.4717(s)	4.2252(s)
	40	5.1752(s)	7.768 (s)	7.778(s)	4.70135(s)	5.4601(s)	6.03209(s)	3.2979(s)	3.2771(s)	4.255(s)
	50	4.8055 (s)	9.6696(s)	9.583(s)	4.6983(s)	5.0036 (s)	6.2935(s)	5.554021(s)	5.4608(s)	5.669(s)
	60	12.133 (s)	11.8221 (s)	12.084 (s)	10.0858 (s)	8.5581 (s)	8.554 (s)	7.3318(s)	8.0564(s)	8.6832(s)
	70	12.4937 (s)	12.1841(s)	12.4239(s)	10.4135(s)	9.0759 (s)	9.1809(s)	7.5998 (s)	8.6841 (s)	9.01558(s)
	80	15.04687 (s)	14.3992 (s)	14.099 (s)	12.5666 (s)	12.3373 (s)	12.277(s)	9.8387 (s)	10.5093 (s)	10.308(s)

Effectiveness of Collector Fog Nodes: The results in Table 9.6 show that when the number of CFNs is 50 or less the average response time decreases. When the number of CFNs is 60 or higher the average response time starts increasing. A similar pattern is seen in Table 9.7 where the number of vehicles is 2000.

Effectiveness of Algorithms 8.1 and 8.2: From Table 9.6 and 9.7 we see that the original query graph results in smaller response times compared to the reconfigured and the adjusted query graph. The response time of the reconfigured and adjusted query graphs is significantly higher for a small number of CFNs. However, when the number of CFNs increases the differences decreases.

Effectiveness of Fog-Tree Hierarchy: By comparing the results in Tables 9.6 and 9.7, we can conclude that the Tree 3, representing the shortest tree, has the lowest response time as the result of providing more fog nodes closer to data sources than the other two trees. In Table 9.7, when the number of vehicles was doubled, we did not observe a significant change in the average response time when the number of level 0 fog nodes was sufficient. As the results show, Tree 3 (which is a shortest tree among Tree 1 and Tree 2) has the lowest response time. The explanation is that Tree 3 provides more processing power closer to the data sources and this results in lower execution time.

Comparison of Cost Functions C_1, C_2, and C_3: From Table 9.6 and 9.7 we see that cost function C_3 has lower response time compare to cost functions C_1 and C_2. The explantation is

Table 9.7: Average Response Time-2000 Vehicles

		Tree 1			Tree 2			Tree 3		
	$\#CFNs$	Original QG	Reconfigured QG	Adjusted QG	Original QG	Reconfigured QG	Adjusted QG	Original QG	Reconfigured QG	Adjusted QG
C_1	2	36.088465 (s)	69.20633 (s)	69.97032 (s)	35.32318 (s)	68.5676 (s)	67.45029 (s)	31.305846 (s)	12.27104 (s)	59.07281 (s)
	10	10.17136 (s)	15.20343 (s)	15.49831 (s)	9.02956 (s)	14.11579 (s)	14.087428 (s)	8.286048 (s)	12.248578 (s)	12.23371 (s)
	20	8.24454 (s)	9.24454 (s)	9.61137 (s)	7.083165 (s)	6.49282 (s)	6.866296 (s)	6.03985 (s)	7.00221 (s)	6.69465 (s)
	30	7.10888 (s)	8.144731 (s)	8.532131 (s)	6.072502 (s)	5.4297 (s)	5.785701 (s)	5.074091 (s)	6.025531 (s)	5.702771 (s)
	40	7.968731 (s)	9.053891 (s)	9.451935 (s)	6.410475 (s)	5.8557 (s)	6.213302 (s)	5.053103 (s)	5.99392 (s)	5.670013 (s)
	50	12.82133 (s)	11.61925 (s)	11.6871 (s)	10.70726 (s)	10.76323 (s)	10.03823 (s)	8.923274 (s)	8.56548 (s)	8.87268 (s)
	60	13.96103 (s)	14.042763 (s)	14.912609 (s)	12.054752 (s)	10.49328 (s)	10.8404 (s)	10.078331 (s)	9.260107 (s)	9.10067 (s)
	70	14.98873 (s)	15.108898 (s)	16.001572 (s)	13.154133 (s)	11.545703 (s)	11.86181 (s)	11.0099(s)	9.96187 (s)	9.778313 (s)
	80	17.9942 (s)	17.8725 (s)	17.92251 (s)	14.333492 (s)	15.3769 (s)	15.200854 (s)	12.27104 (s)	12.27709 (s)	12.3629 (s)
C_2	2	32.5031 (s)	62.30884(s)	62.4011 (s)	32.02837(s)	61.81492(s)	60.7879(s)	28.3208(s)	53.3063(s)	53.276(s)
	10	9.1623 (s)	13.6899(s)	13.9549 (s)	8.1347(s)	12.7278(s)	12.6967(s)	7.4964(s)	11.047(s)	11.032 (s)
	20	7.4237(s)	8.3231(s)	8.6529(s)	6.3835(s)	5.8566 (s)	6.19(s)	5.456(s)	6.3167(s)	6.0388 (s)
	30	6.40115(s)	7.3327(s)	7.6811(s)	5.4731(s)	4.8969(s)	5.2163(s)	4.5883(s)	5.43727(s)	5.14586 (s)
	40	7.1749(s)	8.1507(s)	8.50896(s)	5.782(s)	5.2796(s)	5.601(s)	4.5694(s)	5.41185(s)	5.1151(s)
	50	11.54221 (s)	10.459 (s)	10.52007 (s)	9.6554(s)	9.6962(s)	9.04246(s)	8.052154 (s)	7.7205(s)	7.99746(s)
	60	12.5686(s)	12.64129(s)	13.42378(s)	10.8687(s)	9.4574(s)	9.7684(s)	9.093(s)	8.2626(s)	8.2074 (s)
	70	13.4874(s)	13.6008(s)	14.40415(s)	11.86102 (s)	10.4066(s)	10.685(s)	9.9477(s)	8.994(s)	8.819(s)
	80	16.2025(s)	16.0922(s)	16.13638(s)	12.9284(s)	13.8562(s)	13.6948 (s)	11.1201(s)	11.1099 (s)	11.1519(s)
C_3	2	31.0662 (s)	59.54709(s)	59.63514(s)	30.641107(s)	59.10205(s)	58.11314 (s)	27.1092(s)	50.973 (s)	50.948(s)
	10	8.75776 (s)	13.0837 (s)	13.3368(s)	7.7759 (s)	12.169 (s)	12.1383 (s)	7.17604(s)	10.5639(s)	10.549 (s)
	20	7.09503(s)	7.954215(s)	8.2692(s)	6.1026(s)	5.60062(s)	5.91825(s)	5.2208 (s)	6.0406(s)	5.7748 (s)
	30	6.11768 (s)	7.00763 (s)	7.3405(s)	5.2328(s)	4.6827(s)	4.9878(s)	4.3908(s)	5.20021(s)	4.9215(s)
	40	6.857(s)	7.78917 (s)	8.1315(s)	5.53001(s)	5.048 (s)	5.35504(s)	4.37272(s)	5.1755(s)	4.8926(s)
	50	11.0301 (s)	9.9957(s)	10.0535 (s)	9.2324(s)	9.2683(s)	8.6432 (s)	7.70118(s)	7.3834(s)	7.6459(s)
	60	12.0108 (s)	12.0803(s)	12.8279 (s)	10.3919(s)	9.04155(s)	9.33804 (s)	8.696(s)	7.901515(s)	7.8481(s)
	70	12.8892(s)	12.9973 (s)	13.7648 (s)	11.34212 (s)	9.9497(s)	10.217(s)	9.5181(s)	8.6072(s)	8.4332(s)
	80	15.4853(s)	15.379(s)	15.42131 (s)	12.3634(s)	13.2484(s)	13.2901(s)	10.6506(s)	10.8341 (s)	10.8635(s)

that cost function C_3 considers both execution time and network latency at the same time for mapping vertices of query graph to find a mapping that provide lower response time.

9.2.4 Average Distortion Due to Buffering

Tables 9.8 and 9.9 show the average distortion due to buffering for processing of the original query graph, reconfigured query graph, and adjusted query graph as the number of fog nodes and vehicles vary. As Tables 9.8 and 9.9 show the distortion due to buffering is zero.

9.2.5 Average Distortion Due to Network Latency

Tables 9.10 and 9.11 show the average distortion due to buffering for processing of the original query graph, reconfigured query graph, and adjusted query graph as the number of fog nodes and vehicles vary.

Tables 9.10 and 9.11 show that for a large number of vehicles the average distortion due to network latency decreases by increasing the number of level 0 fog nodes. However, increasing the number of level 0 fog nodes will lead to an increase in end-to-end delay and increasing the end-to-end delay results in an increase of the average distortion due to network latency. We experience more network delay because an increase in the number of fog nodes at level 0 results in generating more results for upper levels fog nodes. Furthermore, as Tables 9.10 and 9.11 show processing the original query graph has a larger distortion compared with the reconfigured and adjusted query graph. The reason is that an original query graph results in the transfer of more data (sometimes redundant data) among query vertices which can lead to an increase in the network latency. However, when we merge query vertices in a configured/adjusted query graph then less data is transferred among query graph vertices.

Table 9.8: Average Distortion Due to Buffering-1000 Vehicles

		Tree 1			Tree 2			Tree 3		
	$\#CFNs$	Original QG	Reconfigured QG	Adjusted QG	Original QG	Reconfigured QG	Adjusted QG	Original QG	Reconfigured QG	Adjusted QG
C_1	2	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	10	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	20	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	30	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	40	0 (%)	0 (%)	0 (s)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	50	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	60	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	70	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	80	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
C_2	2	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	10	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	20	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	30	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	40	0 (%)	0 (%)	0 (s)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	50	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	60	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	70	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	80	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
C_3	2	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	10	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	20	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	30	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	40	0 (%)	0 (%)	0 (s)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	50	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	60	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	70	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	80	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)

Table 9.9: Average Distortion Due to Buffering-2000 Vehicles

		Tree 1			Tree 2			Tree 3		
	$\#CFNs$	Original QG	Reconfigured QG	Adjusted QG	Original QG	Reconfigured QG	Adjusted QG	Original QG	Reconfigured QG	Adjusted QG
C_1	2	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	10	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	20	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	30	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	40	0 (%)	0 (%)	0 (s)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	50	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	60	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	70	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	80	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
C_2	2	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	10	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	20	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	30	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	40	0 (%)	0 (%)	0 (s)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	50	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	60	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	70	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	80	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
C_3	2	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	10	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	20	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	30	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	40	0 (%)	0 (%)	0 (s)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	50	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	60	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	70	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	80	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)

9.3 Camera Crowd Size Measurement Scenario Results

This section presents the results for the Camera Crowd Size Scenario. The information about details of query operators that are used in the query graph can be found in Appendix B.

Table 9.12 shows the embedding distortion for mapping the query graph that is used by camera crowd size measurement scenario. Distortion values that are closer to 1 means that query operators are placed closer to the data sources than mappings that result in lower distor-

Table 9.10: Average Distortion Due to Network Latency-1000 Vehicles

		Tree 1			Tree 2			Tree 3		
	$\#CFNs$	Original QG	Reconfigured QG	Adjusted QG	Original QG	Reconfigured QG	Adjusted QG	Original QG	Reconfigured QG	Adjusted QG
C_1	2	9.68 (%)	9.53 (%)	9.57 (%)	40.12 (%)	39.25 (%)	36.79 (%)	41.63 (%)	39.58 (%)	43.61 (%)
	10	0 (%)	0 (%)	0 (%)	0 (%)	10.65 (%)	0.69 (%)	0 (%)	11.03 (%)	7.55 (%)
	20	0 (%)	0 (%)	0 (%)	0(%)	0 (%)	0(%)	13.61 (%)	0 (%)	0 (%)
	30	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	15.24 (%)	0.55 (%)	0 (%)
	40	0 (%)	0 (%)	0 (%)	1.15 (%)	0 (%)	0 (%)	19.57 (%)	2.96 (%)	0 (%)
	50	0 (%)	0 (%)	0 (%)	19.69 (%)	0 (%)	0 (%)	22.52 (%)	8.07 (%)	2.53 (%)
	60	0 (%)	0 (%)	0 (%)	20.23 (%)	0 (%)	0 (%)	25.6 (%)	12.25 (%)	6.18 (%)
	70	0 (%)	0 (%)	0 (%)	28.18 (%)	25.05 (%)	0 (%)	34.85 (%)	17.34 (%)	12.89 (%)
	80	0 (%)	0 (%)	0 (%)	29.98 (%)	19.59 (%)	0 (%)	35.14 (%)	29.67 (%)	27.58 (%)
C_2	2	14.56 (%)	14.55 (%)	14.55 (%)	42.89 (%)	42.12 (%)	39.9 (%)	44.24 (%)	42.41 (%)	46.01(%)
	10	0 (%)	0 (%)	0 (%)	0 (%)	16.35 (%)	7.37 (%)	0 (%)	16.69 (%)	13.56 (%)
	20	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	18.97 (%)	3.71 (%)	0 (%)
	30	0 (%)	0(%)	0(%)	4.78(%)	0 (%)	0 (%)	22.11 (%)	6.35 (%)	0 (%)
	40	0 (%)	0(%)	0(%)	7.79 (%)	0 (%)	0 (%)	24.38 (%)	9.42 (%)	0 (%)
	50	0 (%)	0(%)	0(%)	24.48 (%)	0 (%)	0 (%)	27.02 (%)	14.01 (%)	9.02 (%)
	60	0 (%)	0(%)	0(%)	24.97 (%)	0 (%)	0 (%)	29.57 (%)	17.7 (%)	12.32 (%)
	70	0 (%)	0(%)	0(%)	32.12 (%)	29.33 (%)	0 (%)	38.15 (%)	32.09 (%)	18.37 (%)
	80	0 (%)	0 (%)	0 (%)	33.77 (%)	24.4 (%)	0 (%)	38.33 (%)	33.46 (%)	31.61 (%)
C_3	2	15.79 (%)	15.68 (%)	15.71 (%)	43.14 (%)	42.39 (%)	40.25 (%)	44.46 (%)	42.68 (%)	46.17 (%)
	10	0 (%)	0 (%)	0 (%)	0 (%)	17.52 (%)	8.85 (%)	0 (%)	17.85 (%)	13.93 (%)
	20	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	20.16 (%)	5.33 (%)	0 (%)
	30	0 (%)	0 (%)	0 (%)	6.4 (%)	0 (%)	0 (%)	23.25 (%)	7.87 (%)	0 (%)
	40	0 (%)	0 (%)	0 (%)	9.26 (%)	0 (%)	0 (%)	25.25 (%)	10.83 (%)	0.54 (%)
	50	0 (%)	0 (%)	0 (%)	25.39 (%)	0 (%)	0 (%)	27.84 (%)	15.27 (%)	9.26 (%)
	60	0 (%)	0 (%)	0 (%)	25.84 (%)	0 (%)	0 (%)	30.52 (%)	18.91 (%)	13.64 (%)
	70	0 (%)	0 (%)	0 (%)	32.77 (%)	30.04 (%)	0 (%)	38.56 (%)	32.71 (%)	19.47 (%)
	80	0 (%)	0 (%)	0 (%)	34.34(%)	25.3 (%)	0 (%)	38.82 (%)	34.06 (%)	32.23 (%)

Table 9.11: Average Distortion Due to Network Latency-2000 Vehicles

		Tree 1			Tree 2			Tree 3		
	$\#CFNs$	Original QG	Reconfigured QG	Adjusted QG	Original QG	Reconfigured QG	Adjusted QG	Original QG	Reconfigured QG	Adjusted QG
C_1	2	30.26 (%)	29.95 (%)	29.64 (%)	45.82 (%)	45.61 (%)	44.33 (%)	46.81 (%)	45.78 (%)	45.91 (%)
	10	0 (%)	0 (%)	0 (%)	0 (%)	29.8 (%)	24.3 (%)	38.11 (%)	30.29 (%)	42.99 (%)
	20	0 (%)	0 (%)	0 (%)	0 (%)	14.52 (%)	5.1 (%)	27.64 (%)	17.57 (%)	16.01 (%)
	30	0 (%)	0 (%)	0 (%)	0 (%)	6.32 (%)	1.18 (%)	28.06 (%)	17.43 (%)	15.28 (%)
	40	0 (%)	0 (%)	0 (%)	15.01 (%)	0 (%)	0 (%)	28.06 (%)	17.33 (%)	13.76 (%)
	50	0 (%)	0 (%)	0 (%)	25.51 (%)	0.39 (%)	0 (%)	28.36 (%)	16.03 (%)	11.51 (%)
	60	0 (%)	0 (%)	0 (%)	26.13 (%)	15.7 (%)	10.51 (%)	29.43 (%)	40.82 (%)	22.47 (%)
	70	0 (%)	0 (%)	0 (%)	29.89 (%)	20.22 (%)	13.73 (%)	38.01 (%)	35.17 (%)	25.13 (%)
	80	0 (%)	0 (%)	0 (%)	33.61 (%)	23.62 (%)	16.93 (%)	43.9 (%)	42.18 (%)	31.51 (%)
C_2	2	33.82 (%)	33.56 (%)	33.31 (%)	46.57 (%)	46.4 (%)	45.39 (%)	47.38 (%)	46.54 (%)	46.65 (%)
	10	3.13 (%)	0 (%)	0 (%)	3.5 (%)	33.44 (%)	28.93 (%)	40.26 (%)	33.84 (%)	33.19 (%)
	20	0 (%)	0 (%)	0 (%)	6.24 (%)	20.92 (%)	13.19 (%)	31.67 (%)	23.42 (%)	22.13 (%)
	30	0 (%)	0 (%)	0 (%)	3.49 (%)	14.19 (%)	9.98 (%)	32.01 (%)	23.3 (%)	21.54 (%)
	40	0 (%)	0 (%)	0 (%)	21.34 (%)	8.98 (%)	8.08 (%)	32.02 (%)	23.22 (%)	20.29 (%)
	50	0 (%)	0 (%)	0 (%)	29.93 (%)	9.33 (%)	2.91 (%)	32.06 (%)	22.13 (%)	18.45 (%)
	60	0 (%)	0 (%)	0 (%)	30.25 (%)	21.88 (%)	17.63 (%)	33.1 (%)	29.54 (%)	27.43 (%)
	70	0 (%)	0 (%)	0 (%)	33.52 (%)	25.59 (%)	20.27 (%)	40.17 (%)	37.85 (%)	29.61 (%)
	80	0.08 (%)	0 (%)	0 (%)	36.56 (%)	28.37 (%)	22.89 (%)	45.07 (%)	43.68 (%)	34.84 (%)
C_3	2	34.45 (%)	34.21 (%)	33.97 (%)	46.71 (%)	46.54 (%)	45.57 (%)	47.48 (%)	46.68 (%)	46.78 (%)
	10	4.97 (%)	0 (%)	0 (%)	5.32 (%)	34.09 (%)	29.76 (%)	40.64 (%)	34.48 (%)	33.85 (%)
	20	0 (%)	0 (%)	0 (%)	7.96 (%)	22.06 (%)	14.64 (%)	32.39(%)	24.46 (%)	23.22 (%)
	30	0 (%)	0 (%)	0 (%)	5.21 (%)	15.6 (%)	11.55 (%)	32.72 (%)	24.36 (%)	22.66 (%)
	40	0 (%)	0 (%)	0 (%)	22.47 (%)	10.59 (%)	9.65 (%)	32.72 (%)	24.28 (%)	21.46 (%)
	50	0 (%)	0 (%)	0 (%)	30.72 (%)	10.93 (%)	4.75 (%)	32.77 (%)	23.22 (%)	19.69 (%)
	60	0 (%)	0 (%)	0 (%)	31.2 (%)	22.99 (%)	18.9 (%)	33.76 (%)	30.35 (%)	28.32 (%)
	70	0 (%)	0 (%)	0 (%)	34.17 (%)	26.55 (%)	21.43 (%)	40.56 (%)	38.32 (%)	30.42 (%)
	80	2.05 (%)	0.43 (%)	0 (%)	37.09 (%)	29.22 (%)	23.96 (%)	45.2 (%)	43.92 (%)	3544 (%)

tion values.

9.3.1 Average Execution Time

Tables 9.13, 9.14, 9.15, 9.16, and 9.17 show the average execution time needed to process the original query graph, reconfigured query graph, and adjusted query graph for the cost functions as the number of fog nodes and cameras vary as well as for the different types of trees.

Table 9.12: Maximum Embedding Distortion for Camera Crowd Size Measurement Scenario

	Tree 1			Tree 2			Tree 3		
	Original QG	Reconfigured QG	Adjusted QG	Original QG	Reconfigured QG	Adjusted QG	Original QG	Reconfigured QG	Adjusted QG
C_1	0.17	0.25	0.25	0.12	0.17	0.17	0.08	0.1	0.1
C_2	0.1	0.12	0.17	0.08	0.1	0.12	0.07	0.07	0.08
C_3	0.08	0.08	0.1	0.08	0.08	0.08	0.07	0.07	0.07

Table 9.13: Average Execution Time-64 Cameras

		Tree 1			Tree 2			Tree 3		
	#CFNs	Original QG	Reconfigured QG	Adjusted QG	Original QG	Reconfigured QG	Adjusted QG	Original QG	Reconfigured QG	Adjusted QG
C_1	2	929.925 (s)	8044.054 (s)	8045.111 (s)	891.303 (s)	7824.061 (s)	7826.948 (s)	726.974 (s)	6896.109 (s)	6898.166 (s)
	10	16.831 (s)	1289.147 (s)	1291.109 (s)	16.328 (s)	1241.752 (s)	1243.166 (s)	14.508 (s)	1038.084 (s)	1049.042 (s)
	20	10.688 (s)	480.591 (s)	480.898 (s)	10.442 (s)	454.834 (s)	463.034 (s)	9.221 (s)	336.102 (s)	337.833 (s)
	30	10.538 (s)	40.283 (s)	50.382 (s)	9.834 (s)	27.891 (s)	30.401 (s)	7.531 (s)	23.906 (s)	24.112 (s)
	40	10.012 (s)	32.601 (s)	36.481 (s)	9.321 (s)	27.02 (s)	27.842 (s)	7.344 (s)	23.838 (s)	24.1 (s)
	50	9.029 (s)	28.995 (s)	29.395 (s)	8.413 (s)	27.011 (s)	27.362 (s)	7.339 (s)	24.065 (s)	24.165 (s)
	60	9.006 (s)	28.018 (s)	29.133 (s)	8.014 (s)	26.141 (s)	27.143 (s)	7.029 (s)	23.046 (s)	24.022 (s)
	70	9.221 (s)	27.806 (s)	28.414 (s)	8.421 (s)	26.136 (s)	26.529 (s)	7.311 (s)	19.033 (s)	19.872 (s)
	80	9.455 (s)	17.003 (s)	17.198 (s)	8.44 (s)	15.001 (s)	15.833 (s)	7.301 (s)	13.941 (s)	14.491 (s)
C_2	2	910.28 (s)	718.957(s)	721.805 (s)	873.628(s)	709.051(s)	711.052(s)	711.853(s)	625.083(s)	621.682 (s)
	10	16.976(s)	109.823(s)	112.139(s)	17.073 (s)	113.48454 (s)	114.616(s)	15.295 (s)	91.056(s)	93.4741 (s)
	20	11.603 (s)	40.593 (s)	40.4758(s)	11.3188(s)	38.267(s)	39.559 (s)	10.125(s)	28.541(s)	29.695(s)
	30	11.404(s)	5.808 (s)	5.998(s)	10.724(s)	3.654 (s)	3.9736(s)	8.473(s)	3.293(s)	3.812 (s)
	40	10.95 (s)	4.017(s)	4.084(s)	10.214(s)	3.575(s)	3.867907 (s)	8.29(s)	3.287 (s)	3.99(s)
	50	9.501(s)	3.84(s)	3.909(s)	9.335(s)	3.573(s)	3.857(s)	8.285 (s)	3.308(s)	3.863(s)
	60	9.474(s)	3.668(s)	3.685(s)	8.945(s)	3.496 (s)	3.604(s)	7.982(s)	3.215(s)	3.762(s)
	70	9.385(s)	3.65(s)	3.674(s)	9.34351(s)	3.494(s)	3.599(s)	8.258(s)	2.852(s)	3.74(s)
	80	9.225 (s)	2.673(s)	2.681(s)	9.356(s)	2.511(s)	2.564(s)	8.312(s)	2.395(s)	3.397(s)
C_3	2	819.334 (s)	644.523 (s)	632.608 (s)	786.344(s)	626.924(s)	623.184(s)	640.732(s)	552.682 (s)	544.858 (s)
	10	15.28(s)	104.176(s)	98.282(s)	15.368 (s)	100.34 (s)	100.453(s)	13.767 (s)	84.046 (s)	81.923 (s)
	20	10.444 (s)	39.428 (s)	35.474(s)	10.188(s)	37.372(s)	32.042 (s)	9.114(s)	27.888 (s)	26.026 (s)
	30	10.274(s)	5.224 (s)	4.0306 (s)	9.653(s)	3.23128 (s)	3.132(s)	7.627(s)	2.912(s)	2.728 (s)
	40	9.856 (s)	3.608(s)	3.518(s)	9.202(s)	3.161(s)	3.127 (s)	7.462(s)	2.907 (s)	2.628(s)
	50	8.552(s)	3.396 (s)	3.251(s)	8.403(s)	3.16(s)	3.118(s)	7.458 (s)	2.925(s)	2.51(s)
	60	8.528(s)	3.244(s)	3.204 (s)	8.052(s)	3.0918 (s)	3.071(s)	7.185(s)	2.843(s)	2.421(s)
	70	8.448(s)	3.228(s)	3.212 (s)	8.41(s)	3.09 (s)	2.822 (s)	7.433(s)	2.522 (s)	2.409 (s)
	80	8.304 (s)	2.364(s)	2.324(s)	8.422(s)	2.221(s)	2.204(s)	7.48(s)	2.118 (s)	2.101(s)

Table 9.14: Average Execution Time-128 Cameras

		Tree 1			Tree 2			Tree 3		
	#CFNs	Original QG	Reconfigured QG	Adjusted QG	Original QG	Reconfigured QG	Adjusted QG	Original QG	Reconfigured QG	Adjusted QG
C_1	2	1143.807 (s)	10860.899 (s)	11181.235 (s)	1096.302 (s)	10566.379 (s)	10718.963 (s)	894.178 (s)	9312.524 (s)	9447.669 (s)
	10	21.702 (s)	1742.997 (s)	1769.131 (s)	20.083 (s)	1678.241 (s)	1721.224 (s)	17.844 (s)	1404.056 (s)	1432.175 (s)
	20	14.146 (s)	648.797 (s)	658.83 (s)	12.843 (s)	614.259 (s)	622.058 (s)	11.383 (s)	456.074 (s)	460.974 (s)
	30	13.961 (s)	57.382 (s)	71.023 (s)	12.695 (s)	37.566 (s)	43.045 (s)	10.263 (s)	32.273 (s)	33.033 (s)
	40	13.314 (s)	44.011 (s)	50.978 (s)	12.464 (s)	38.017 (s)	38.587 (s)	10.033 (s)	32.181 (s)	33.017 (s)
	50	12.105 (s)	40.683 (s)	42.723 (s)	12.347 (s)	36.93 (s)	37.128 (s)	10.026 (s)	32.622 (s)	32.969 (s)
	60	12.077 (s)	39.384 (s)	39.729 (s)	11.857 (s)	36.508 (s)	36.635 (s)	9.64 (s)	31.573 (s)	32.429 (s)
	70	12.341 (s)	39.094 (s)	39.358 (s)	11.357 (s)	26.529 (s)	35.321 (s)	9.992 (s)	26.075 (s)	26.827 (s)
	80	12.629 (s)	24.217 (s)	24.294 (s)	11.301 (s)	21.551 (s)	23.374 (s)	9.98 (s)	19.062 (s)	19.599 (s)
C_2	2	1124.53(s)	1013.09(s)	1026.7252 (s)	1077.921(s)	961.281(s)	977.448 (s)	879.594(s)	852.539 (s)	895.059 (s)
	10	23.523(s)	161.923(s)	162.926(s)	21.935(s)	152.7312(s)	158.55 (s)	19.738(s)	121.61(s)	124.183 (s)
	20	16.109(s)	56.621(s)	57.551 (s)	14.831 (s)	56.988 (s)	58.752 (s)	13.398 (s)	40.29 (s)	41.51(s)
	30	16.243(s)	7.958 (s)	8.582(s)	14.686(s)	6.123 (s)	6.385(s)	12.299 (s)	5.335(s)	5.988(s)
	40	15.311(s)	7.678(s)	7.748(s)	14.419 (s)	5.7217 (s)	6.533 (s)	12.074(s)	5.189(s)	5.984 (s)
	50	14.075(s)	5.704 (s)	5.77(s)	14.312(s)	5.743 (s)	5.964(s)	12.06(s)	5.368 (s)	5.962(s)
	60	14.24(s)	5.23 (s)	5.689(s)	13.866(s)	5.74 (s)	5.965(s)	11.688 (s)	5.135(s)	5.927(s)
	70	14.28(s)	5.855 (s)	5.926(s)	13.375(s)	5.248(s)	5.79 (s)	12.037 (s)	4.743(s)	4.971(s)
	80	13.549(s)	4.404(s)	4.472(s)	13.366(s)	4.398(s)	4.4904(s)	12.0226(s)	3.97(s)	3.989 (s)
C_3	2	1008.556(s)	900.525 (s)	816.498 (s)	966.746 (s)	854.472 (s)	799.514 (s)	788.874(s)	757.813 (s)	700.493 (s)
	10	21.097(s)	150.154 (s)	143.548(s)	19.673 (s)	144.65 (s)	139.692 (s)	17.702(s)	114.32(s)	109.413 (s)
	20	14.448 (s)	54.775(s)	50.706 (s)	13.30184 (s)	54.212 (s)	51.764 (s)	12.017 (s)	38.485 (s)	36.573(s)
	30	14.568 (s)	7.0745 (s)	6.681(s)	13.1716 (s)	5.443 (s)	4.745(s)	11.031 (s)	4.743 (s)	4.642(s)
	40	13.732(s)	6.825(s)	5.074(s)	12.932 (s)	5.086 (s)	4.875 (s)	10.829 (s)	4.613 (s)	4.441 (s)
	50	12.624 (s)	5.151 (s)	5.084(s)	12.836 (s)	5.105 (s)	4.97(s)	10.822(s)	4.772 (s)	4.637 (s)
	60	12.776(s)	5.538 (s)	4.132(s)	12.436(s)	5.103 (s)	4.93 (s)	10.483 (s)	4.565(s)	4.394 (s)
	70	12.808(s)	5.205(s)	4.164(s)	11.996(s)	4.665(s)	4.228 (s)	10.796 (s)	4.216(s)	4.116 (s)
	80	12.152 (s)	3.915(s)	3.852(s)	11.988 (s)	3.91 (s)	3.692 (s)	10.7824(s)	3.529 (s)	3.462 (s)

Table 9.15: Average Execution Time-256 Cameras

		Tree 1			Tree 2			Tree 3		
	$\#CFNs$	Original QG	Reconfigured QG	Adjusted QG	Original QG	Reconfigured QG	Adjusted QG	Original QG	Reconfigured QG	Adjusted QG
C_1	2	5519.175 (s)	33977.192 (s)	33980.167 (s)	5364.932 (s)	33107.432 (s)	33107.533 (s)	4657.464 (s)	29381.033 (s)	29388.083 (s)
	10	643.753 (s)	6423.823 (s)	6443.834 (s)	612.33 (s)	6235.441 (s)	6239.231 (s)	478.432 (s)	5491.024 (s)	5491.331 (s)
	20	166.442 (s)	3719.919 (s)	3722.515 (s)	162.311 (s)	3612.252 (s)	3614.166 (s)	64.655 (s)	3144.094 (s)	3148.621 (s)
	30	98.517 (s)	2727.108 (s)	2730.388 (s)	98.221 (s)	2725.927 (s)	2730.144 (s)	18.382 (s)	2045.442 (s)	2066.726 (s)
	40	90.212 (s)	1819.972 (s)	1821.737 (s)	90.198 (s)	1817.021 (s)	1817.378 (s)	16.482 (s)	1541.303 (s)	1552.261 (s)
	50	17.997 (s)	1017.833 (s)	1020.626 (s)	17.533 (s)	979.275 (s)	979.621 (s)	15.643 (s)	802.266 (s)	805.112 (s)
	60	17.097 (s)	1009.356 (s)	1011.833 (s)	17.414 (s)	965.914 (s)	966.013 (s)	14.932 (s)	790.732 (s)	793.028 (s)
	70	16.442 (s)	491.938 (s)	492.772 (s)	14.892 (s)	464.532 (s)	466.341 (s)	13.234 (s)	338.03 (s)	341.293 (s)
	80	16.099 (s)	477.992 (s)	480.393 (s)	14.874 (s)	452.802 (s)	453.157 (s)	13.114 (s)	336.932 (s)	337.633 (s)
C_2	2	5029.347(s)	2892.302 (s)	2899.421(s)	4888.791(s)	2487.9 (s)	2499.609 (s)	4244.11 (s)	2387.557 (s)	2397.773(s)
	10	586.612(s)	546.757 (s)	580.717 (s)	557.696 (s)	460.021 (s)	469.164(s)	435.97 (s)	467.422(s)	466.372(s)
	20	151.67 (s)	316.579 (s)	377.052 (s)	147.904(s)	307.493 (s)	318.983 (s)	58.942 (s)	267.64(s)	278.937(s)
	30	89.771(s)	201.127 (s)	209.465(s)	89.503 (s)	201.104(s)	207.2 (s)	16.742 (s)	174.118(s)	179.464(s)
	40	82.206(s)	134.268(s)	135.15 (s)	82.192(s)	154.672 (s)	155.253 (s)	16.146 (s)	131.202 (s)	135.49 (s)
	50	20.876(s)	86.435 (s)	86.195(s)	17.103(s)	83.357 (s)	82.907 (s)	15.944(s)	59.186 (s)	58.801 (s)
	60	20.077(s)	85.815 (s)	85.438 (s)	16.993(s)	82.23 (s)	71.894(s)	15.855(s)	58.336 (s)	56.651(s)
	70	18.351(s)	41.96(s)	36.593(s)	16.944(s)	34.274 (s)	34.3242 (s)	15.633(s)	24.938(s)	23.919 (s)
	80	18.0528(s)	35.263(s)	34.383(s)	16.924(s)	31.406 (s)	32.042(s)	15.088 (s)	19.454 (s)	20.779 (s)
C_3	2	4470.531(s)	2548.284 (s)	2523.512(s)	4345.592(s)	2191.983 (s)	2183.065 (s)	3772.544 (s)	2103.575 (s)	2094.125 (s)
	10	521.433(s)	481.725 (s)	419.841 (s)	495.73 (s)	405.305 (s)	383.55 (s)	387.529 (s)	411.826 (s)	389.845(s)
	20	134.818 (s)	278.925 (s)	241.9675 (s)	131.471(s)	270.919 (s)	234.92 (s)	52.375 (s)	235.807 (s)	226.146(s)
	30	79.797(s)	177.202 (s)	165.472 (s)	79.559 (s)	177.185 (s)	154.76 (s)	14.882 (s)	153.408 (s)	148.004 (s)
	40	73.072(s)	118.298 (s)	115.415 (s)	73.06(s)	136.275 (s)	118.1257 (s)	14.352 (s)	115.597 (s)	100.865 (s)
	50	18.557(s)	76.155 (s)	66.546(s)	15.203 (s)	73.4425 (s)	63.675 (s)	14.173 (s)	52.147 (s)	51.355 (s)
	60	17.847(s)	75.608 (s)	65.885 (s)	15.105(s)	72.45 (s)	62.790(s)	14.094(s)	51.398 (s)	49.477(s)
	70	16.312 (s)	36.97 (s)	31.959 (s)	15.062(s)	30.198 (s)	29.9775 (s)	13.719(s)	21.972(s)	20.89 (s)
	80	16.047 (s)	31.069 (s)	30.029(s)	15.044(s)	29.433 (s)	27.985(s)	13.234 (s)	18.903 (s)	17.275 (s)

Table 9.16: Average Execution Time-512 Cameras

		Tree 1			Tree 2			Tree 3		
	$\#CFNs$	Original QG	Reconfigured QG	Adjusted QG	Original QG	Reconfigured QG	Adjusted QG	Original QG	Reconfigured QG	Adjusted QG
C_1	2	15398.498(s)	95475.909(s)	95144.467(s)	11749.201(s)	73829.573(s)	75154.099(s)	9827.249(s)	64050.651(s)	63772.14 (s)
	10	1796.07 (s)	18050.942 (s)	18042.735 (s)	1341.002 (s)	13905.033 (s)	14163.054 (s)	1009.491(s)	11970.432(s)	11916.188 (s)
	20	464.373 (s)	10452.972(s)	10423.042(s)	355.461 (s)	8163.689 (s)	8204.156 (s)	136.422(s)	6854.124(s)	7147.369 (s)
	30	274.862(s)	7663.173(s)	7781.605(s)	215.103 (s)	6160.595 (s)	6197.426 (s)	38.786(s)	4459.063 (s)	4484.795 (s)
	40	251.691 (s)	5114.121(s)	5100.863(s)	197.533 (s)	4124.637 (s)	4307.185 (s)	34.777(s)	3360.04 (s)	3368.406 (s)
	50	50.211(s)	2860.117(s)	2857.7528(s)	38.397 (s)	2222.955 (s)	2223.739 (s)	33.00673(s)	1748.939(s)	1747.093 (s)
	60	47.706 (s)	2836.29(s)	2833.132(s)	38.136 (s)	2192.624 (s)	2192.849 (s)	31.506(s)	1723.795 (s)	1720.87 (s)
	70	45.873 (s)	1382.345(s)	1379.761(s)	32.613 (s)	1049.842 (s)	1058.594 (s)	27.923(s)	736.91 (s)	740.605 (s)
	80	44.916 (s)	1343.157(s)	1345.104(s)	32.574 (s)	1032.388 (s)	1037.729 (s)	27.67(s)	734.511 (s)	732.663(s)
C_2	2	12286.538(s)	7012.595(s)	7100.511(s)	9374.745(s)	4226.742(s)	4353.175(s)	7841.211(s)	3666.899(s)	3710.9(s)
	10	1433.092(s)	1365.757 (s)	1450.361(s)	1069.99 (s)	910.566 (s)	965.999(s)	805.472(s)	799.545(s)	800.291 (s)
	20	370.525(s)	658.275(s)	691.31(s)	283.624(s)	467.372 (s)	464.56(s)	108.325(s)	392.398(s)	379.812(s)
	30	219.31(s)	482.587(s)	481.514(s)	171.43(s)	352.45(s)	368.634(s)	30.946(s)	254.393(s)	253.039 (s)
	40	200.825 (s)	322.054(s)	337.988(s)	157.612 (s)	236.139(s)	237.62 (s)	27.748(s)	230.834(s)	224.19 (s)
	50	40.063(s)	180.115(s)	183.397(s)	32.637(s)	127.266 (s)	120.609(s)	26.335(s)	120.151(s)	119.916(s)
	60	38.059 (s)	144.264(s)	147.034(s)	32.429 (s)	125.527(s)	117.313(s)	25.139(s)	72.35(s)	66.126 (s)
	70	36.445(s)	64.153(s)	60.966(s)	31.022(s)	53.104 (s)	51.341(s)	22.27(s)	42.187(s)	39.773(s)
	80	35.838(s)	51.684(s)	51.565 (s)	28.744(s)	51.102(s)	50.269(s)	22.078(s)	29.515(s)	28.137(s)
C_3	2	10825.144(s)	5251.175(s)	5108.668(s)	8259.688(s)	3691.478(s)	3509.243(s)	6908.556(s)	3202.532(s)	3126.328(s)
	10	1262.637(s)	1192.801 (s)	1082.564(s)	942.724 (s)	795.255 (s)	749.783(s)	709.672(s)	698.521(s)	614.971 (s)
	20	326.454 (s)	574.913(s)	425.38(s)	249.889(s)	408.186 (s)	402.22(s)	95.904(s)	342.706(s)	328.842(s)
	30	193.228(s)	421.474(s)	416.896(s)	151.218(s)	308.021(s)	301.848(s)	27.266(s)	222.178 (s)	219.082 (s)
	40	176.939 (s)	281.27(s)	206.051(s)	138.866 (s)	206.235 (s)	188.416 (s)	24.448(s)	201.602(s)	194.104 (s)
	50	35.298(s)	157.306(s)	141.465(s)	26.993(s)	111.15 (s)	104.424(s)	23.203(s)	104.936(s)	103.824(s)
	60	33.533 (s)	125.995(s)	109.987(s)	26.81 (s)	109.631 (s)	101.57(s)	22.149(s)	63.188 (s)	57.252 (s)
	70	32.111(s)	56.029(s)	52.785(s)	22.927(s)	52.492 (s)	50.512 (s)	19.622(s)	36.845(s)	34.436 (s)
	80	31.576(s)	50.873(s)	50.706 (s)	21.801 (s)	51.618(s)	48.718(s)	19.452(s)	28.398(s)	26.959 (s)

Effectiveness of Collector Fog Nodes: The results in Tables 9.13, 9.14, 9.15, 9.16, and 9.17 show that by increasing the number of CFNs the average execution time decreases for the reconfigured and adjusted query graphs.

Effectiveness Fog-Tree Hierarchy: Tree 3, representing the shortest tree, has the lowest execution time as the result of providing more CFNs closer to data sources than the other two trees. With Tree 2 each fog node can accept up to three children while each fog node in Tree 3 can accept up to four children. By comparing the results in Tables 9.13, 9.14, 9.15, 9.16, and

Table 9.17: Average Execution Time-640 Cameras

		Tree 1			Tree 2			Tree 3		
	#CFNs	Original QG	Reconfigured QG	Adjusted QG	Original QG	Reconfigured QG	Adjusted QG	Original QG	Reconfigured QG	Adjusted QG
C_1	2	17862.257(s)	110752.055(s)	110367.582(s)	13629.073(s)	85642.305(s)	87178.755(s)	11399.608(s)	74298.756(s)	73975.682(s)
	10	2083.442(s)	20939.093 (s)	20929.572(s)	1555.563(s)	16129.837(s)	16429.142 (s)	1171.011(s)	13885.701(s)	13822.772(s)
	20	538.672 (s)	12125.447(s)	12090.728(s)	412.33484 (s)	9469.878 (s)	9516.821 (s)	158.278(s)	7950.772(s)	8290.982 (s)
	30	318.84(s)	8889.281(s)	9026.662(s)	249.52 (s)	7146.29 (s)	7189.008 (s)	44.991(s)	5172.516 (s)	5202.362 (s)
	40	291.968(s)	5932.38(s)	5917.006(s)	229.138(s)	4784.579(s)	4996.356 (s)	40.342(s)	3897.644(s)	3907.352 (s)
	50	58.245(s)	3317.728(s)	3314.99(s)	44.54(s)	2578.623 (s)	2579.538(s)	38.2878(s)	2028.77(s)	2026.627(s)
	60	55.332 (s)	3290.096(s)	3286.434(s)	44.236(s)	2543.444 (s)	2543.705 (s)	36.54732(s)	1999.603(s)	1996.216 (s)
	70	53.212(s)	1603.521(s)	1600.526(s)	37.868 (s)	1217.819(s)	1227.969 (s)	32.394(s)	854.864 (s)	859.106 (s)
	80	52.102(s)	1558.062(s)	1560.314(s)	37.785 (s)	1197.576 (s)	1203.766(s)	32.094(s)	852.033(s)	849.86(s)
C_2	2	13883.787(s)	7924.235(s)	8023.53(s)	10593.465(s)	4776.216(s)	4919.085(s)	8860.563(s)	4143.597(s)	4193.317(s)
	10	1619.396(s)	1543.301 (s)	1638.903(s)	1209.088 (s)	1028.938 (s)	1091.587(s)	910.183(s)	903.485(s)	904.383 (s)
	20	418.695(s)	743.85(s)	781.183(s)	320.49512(s)	528.13 (s)	524.958(s)	122.405(s)	443.409(s)	429.186(s)
	30	247.803(s)	545.321(s)	544.11(s)	193.7159(s)	398.265(s)	416.552(s)	34.968(s)	287.469(s)	285.907 (s)
	40	226.925(s)	363.902(s)	381.924(s)	178.10156 (s)	266.837(s)	268.516(s)	31.324(s)	260.842(s)	253.337 (s)
	50	45.279(s)	203.525(s)	207.231(s)	36.881(s)	143.818 (s)	136.287(s)	29.758(s)	135.773(s)	135.505(s)
	60	43.007 (s)	163.012(s)	166.142(s)	36.647(s)	128.841(s)	132.569(s)	28.407(s)	81.755(s)	74.728(s)
	70	41.185(s)	72.492(s)	68.898(s)	35.056(s)	60.002(s)	58.013(s)	25.1651(s)	47.671(s)	44.949(s)
	80	40.494(s)	58.402(s)	58.265 (s)	32.482(s)	57.746(s)	56.807(s)	24.944(s)	33.355(s)	31.741(s)
C_3	2	12124.161(s)	5881.316(s)	5721.708(s)	9250.85(s)	4134.456(s)	3930.356(s)	7737.582(s)	3586.834(s)	3501.486(s)
	10	1414.154(s)	1335.937 (s)	1212.471(s)	1055.858 (s)	890.66 (s)	839.756(s)	794.834(s)	782.342(s)	688.762(s)
	20	365.628 (s)	643.906(s)	476.425(s)	279.878(s)	457.162 (s)	450.484(s)	107.418(s)	383.832(s)	368.304(s)
	30	216.416(s)	472.058(s)	466.923(s)	169.366(s)	344.982(s)	338.066(s)	30.532(s)	248.836(s)	245.374 (s)
	40	198.178 (s)	315.022(s)	230.777(s)	155.522 (s)	230.982 (s)	211.022 (s)	27.386(s)	225.794(s)	217.398 (s)
	50	39.533(s)	176.182(s)	158.448(s)	30.216(s)	124.488(s)	116.958(s)	25.986(s)	117.522(s)	116.288(s)
	60	37.556 (s)	141.114(s)	123.184(s)	30.022 (s)	122.782(s)	113.754(s)	24.808(s)	70.776 (s)	64.124 (s)
	70	35.964(s)	62.752(s)	59.112(s)	25.674(s)	56.794 (s)	54.574 (s)	21.974(s)	41.264(s)	38.562 (s)
	80	35.365(s)	54.977(s)	54.792(s)	24.412(s)	53.816(s)	51.566(s)	21.784(s)	31.806(s)	30.198(s)

9.17, we can conclude that the fog-tree hierarchy has a good tolerance for processing of a huge volume of data.

Effectiveness of Algorithms 8.1 and 8.2: From Tables 9.13, 9.14, 9.15, 9.16, and 9.17 we see that the original query graph results in less execution time compared to the reconfigured and the adjusted query graph. A fog node is assigned only one node (representing an data operation) of the original graph, while Algorithm 8.1 and Algorithm 8.2 are used to map one or more nodes to a single fog node. The execution time of the reconfigured and adjusted query graphs is significantly higher for a small number of CFNs. However, when the number of CFNs increases the differences decrease.

Comparison of Cost Functions C_1, C_2, and C_3: From Tables 9.13, 9.14, 9.15, 9.16, and 9.17 we see that the use of cost function C_3 has lower execution time compared to cost functions C_1 and C_2. The cost function C_3 favours mapping query vertices of a given query graph to upper levels of the fog-tree hierarchy. These fog nodes have more processing power compared to fog nodes in the lower levels of the fog-tree hierarchy. The results also show that since the cost function C_2 uses more levels than C_1 that more powerful computing power is available resulting in lower execution times.

9.3.2 Average End-to-End Delay

Tables 9.18, 9.19, 9.20, and 9.21 show the average end-to-end delay for the processing of the original query graph, reconfigured query graph, and adjusted query graph.

Effectiveness of Collector Fog Nodes: Tables 9.18, 9.19, 9.20, and 9.21 show that the average end-to-end delay as the number of fog nodes and cameras vary. Tables 9.18, 9.19, 9.20, and 9.21 show that as the number of fog nodes increases the end-to-end delay decreases when the number of cameras increases. When there are fewer fog nodes available then all the generated tuples from cameras are sent to a limited number of fog nodes which leads to a network

Table 9.18: Average End-to-End Delay-64 Cameras

		Tree 1			Tree 2			Tree 3		
	#CFNs	Original QG	Reconfigured QG	Adjusted QG	Original QG	Reconfigured QG	Adjusted QG	Original QG	Reconfigured QG	Adjusted QG
C_1	2	0.867951 (s)	0.458208 (s)	0.433715 (s)	0.903544 (s)	0.472824 (s)	0.45584 (s)	1.606712 (s)	0.599311 (s)	0.59777 (s)
	10	0.299223 (s)	0.107592 (s)	0.102055 (s)	0.309946 (s)	0.108299 (s)	0.10798 (s)	0.369963 (s)	0.142017 (s)	0.13501 (s)
	20	0.202463 (s)	0.072436 (s)	0.06933 (s)	0.228255 (s)	0.078455 (s)	0.075637 (s)	0.233773 (s)	0.081384 (s)	0.081093 (s)
	30	0.172506 (s)	0.072132 (s)	0.067171 (s)	0.225137 (s)	0.078218 (s)	0.073091 (s)	0.218013 (s)	0.080742 (s)	0.080109 (s)
	40	0.137021 (s)	0.068017 (s)	0.067009 (s)	0.179083 (s)	0.078311 (s)	0.075513 (s)	0.199367 (s)	0.080665 (s)	0.080346 (s)
	50	0.097564 (s)	0.066899 (s)	0.065365 (s)	0.112517 (s)	0.078617 (s)	0.077339 (s)	0.175033 (s)	0.080163 (s)	0.080104 (s)
	60	0.106158 (s)	0.074449 (s)	0.073915 (s)	0.123327 (s)	0.079042 (s)	0.078321 (s)	0.188931 (s)	0.086192 (s)	0.086048 (s)
	70	0.081732 (s)	0.074732 (s)	0.073919 (s)	0.118741 (s)	0.090521 (s)	0.090039 (s)	0.182097 (s)	0.091372 (s)	0.090171 (s)
	80	0.065671 (s)	0.075874 (s)	0.074897 (s)	0.115163 (s)	0.092714 (s)	0.092169 (s)	0.149965 (s)	0.094828 (s)	0.093742 (s)
C_2	2	1.170431(s)	0.535507(s)	0.423652 (s)	1.218429 (s)	0.552589 (s)	0.441253 (s)	2.166651132 (s)	0.70041476 (s)	0.5839017 (s)
	10	0.403502 (s)	0.1257427 (s)	0.099687 (s)	0.417962181 (s)	0.126569(s)	0.1054744(s)	0.4988951 (s)	0.165976(s)	0.131877(s)
	20	0.273015(s)	0.084655 (s)	0.06774(s)	0.307806 (s)	0.091695 (s)	0.073882(s)	0.315242(s)	0.095113 (s)	0.079216(s)
	30	0.232624(s)	0.084306 (s)	0.065612(s)	0.303597 (s)	0.0914133(s)	0.071392(s)	0.293903(s)	0.094367(s)	0.0782507(s)
	40	0.184772(s)	0.079494 (s)	0.065459(s)	0.241493(s)	0.091522(s)	0.07376104 (s)	0.268846(s)	0.094278 (s)	0.078417(s)
	50	0.131565(s)	0.078183 (s)	0.063848 (s)	0.1517291 (s)	0.091879(s)	0.075544(s)	0.236032 (s)	0.093686 (s)	0.078257(s)
	60	0.143154(s)	0.087004 (s)	0.0722017 (s)	0.166306 (s)	0.09237(s)	0.076503(s)	0.2547734(s)	0.1007325 (s)	0.084016 (s)
	70	0.110215(s)	0.087339(s)	0.0722047 (s)	0.160122(s)	0.105791(s)	0.0879509(s)	0.245558 (s)	0.106786(s)	0.088079 (s)
	80	0.088557 (s)	0.088673 (s)	0.07315(s)	0.155273 (s)	0.10835 (s)	0.090036 (s)	0.202227 (s)	0.110825(s)	0.091567 (s)
C_3	2	1.3019265 (s)	0.5956704(s)	0.4770865 (s)	1.355316 (s)	0.6146712 (s)	0.501424 (s)	2.410068 (s)	0.7791043 (s)	0.657547 (s)
	10	0.4488345 (s)	0.1398696(s)	0.1122605 (s)	0.464919 (s)	0.1407887 (s)	0.118778(s)	0.554945 (s)	0.184622 (s)	0.148511(s)
	20	0.3036945 (s)	0.094166 (s)	0.076263(s)	0.3423825 (s)	0.1019915 (s)	0.0832007(s)	0.350659(s)	0.1057992 (s)	0.089202(s)
	30	0.258759(s)	0.0937716 (s)	0.073888 (s)	0.3377055 (s)	0.1016834(s)	0.0804001 (s)	0.327015 (s)	0.104966(s)	0.088119(s)
	40	0.2055315 (s)	0.08842 (s)	0.073709 (s)	0.268625(s)	0.1018043(s)	0.0830643 (s)	0.299055 (s)	0.1048645 (s)	0.088306 (s)
	50	0.146346(s)	0.086968 (s)	0.0719015 (s)	0.1687755 (s)	0.1022021(s)	0.0850729 (s)	0.262545 (s)	0.1042119 (s)	0.088114(s)
	60	0.159237 (s)	0.096783 (s)	0.0813065 (s)	0.1849905 (s)	0.102754(s)	0.0861531 (s)	0.28339(s)	0.1120496 (s)	0.094652 (s)
	70	0.122598 (s)	0.097116 (s)	0.081309 (s)	0.178115 (s)	0.117673(s)	0.099029(s)	0.2731455 (s)	0.1187836 (s)	0.099188 (s)
	80	0.09855 (s)	0.098636 (s)	0.082367 (s)	0.1727445 (s)	0.120528 (s)	0.1013859 (s)	0.2249475 (s)	0.1232764 (s)	0.103116 (s)

Table 9.19: Average End-to-End Delay-128 Cameras

		Tree 1			Tree 2			Tree 3		
	#CFNs	Original QG	Reconfigured QG	Adjusted QG	Original QG	Reconfigured QG	Adjusted QG	Original QG	Reconfigured QG	Adjusted QG
C_1	2	1.132861 (s)	0.617775 (s)	0.56845 (s)	1.175217 (s)	0.634291 (s)	0.592307 (s)	2.011987 (s)	0.877221 (s)	0.745591 (s)
	10	0.456075 (s)	0.221578 (s)	0.210219 (s)	0.468835 (s)	0.222377 (s)	0.217698 (s)	0.540255 (s)	0.260479 (s)	0.24581 (s)
	20	0.34093 (s)	0.181852 (s)	0.174876 (s)	0.371623 (s)	0.188654 (s)	0.181687 (s)	0.378189 (s)	0.191963 (s)	0.187584 (s)
	30	0.305282 (s)	0.181509 (s)	0.172544 (s)	0.367913 (s)	0.188386 (s)	0.178938 (s)	0.359436 (s)	0.191238 (s)	0.186517 (s)
	40	0.263054 (s)	0.176859 (s)	0.172369 (s)	0.313108 (s)	0.188491 (s)	0.181554 (s)	0.337246 (s)	0.191151 (s)	0.186773 (s)
	50	0.216101 (s)	0.175595 (s)	0.170594 (s)	0.233895 (s)	0.188837 (s)	0.183526 (s)	0.308289 (s)	0.191843 (s)	0.190584 (s)
	60	0.226328 (s)	0.184127 (s)	0.179828 (s)	0.246759 (s)	0.189317 (s)	0.184586 (s)	0.324827 (s)	0.197396 (s)	0.092931 (s)
	70	0.197261 (s)	0.184447 (s)	0.179832 (s)	0.241301 (s)	0.192288 (s)	0.182421 (s)	0.316695 (s)	0.197636 (s)	0.193712 (s)
	80	0.178148 (s)	0.1857372 (s)	0.180886 (s)	0.237043 (s)	0.204766 (s)	0.199542 (s)	0.278458 (s)	0.207155 (s)	0.202429 (s)
C_2	2	1.731351(s)	0.833069 (s)	0.664347 (s)	1.796084 (s)	0.855341(s)	0.692229(s)	3.074917(s)	1.182931(s)	0.8713722 (s)
	10	0.697019 (s)	0.298797 (s)	0.24568294 (s)	0.71652053(s)	0.299875 (s)	0.2544236(s)	0.82567171 (s)	0.35125(s)	0.2873713 (s)
	20	0.52104331(s)	0.245227(s)	0.204312 (s)	0.567943(s)	0.254399(s)	0.212335(s)	0.577982(s)	0.258821(s)	0.219224(s)
	30	0.46656(s)	0.244768(s)	0.201652(s)	0.56228143(s)	0.2540382 (s)	0.209124(s)	0.549326(s)	0.257884 (s)	0.217984(s)
	40	0.402022(s)	0.238494(s)	0.201446(s)	0.4785229 (s)	0.2541803 (s)	0.212182(s)	0.515411(s)	0.257767 (s)	0.21828(s)
	50	0.33026715 (s)	0.236789(s)	0.1993732(s)	0.357468(s)	0.254646(s)	0.214468(s)	0.47115807(s)	0.258002 (s)	0.222735 (s)
	60	0.3458974 (s)	0.248295(s)	0.2101649 (s)	0.3771217(s)	0.255293(s)	0.215756(s)	0.4964331(s)	0.266188 (s)	0.1086084(s)
	70	0.301473(s)	0.248726 (s)	0.2101695(s)	0.3687803(s)	0.259003(s)	0.213154(s)	0.4840049 (s)	0.266512(s)	0.226391(s)
	80	0.272263(s)	0.25046 (s)	0.211414(s)	0.362272(s)	0.276126951 (s)	0.2332047(s)	0.42556736 (s)	0.279348 (s)	0.236578(s)
C_3	2	1.925863(s)	0.926662 (s)	0.738985 (s)	1.9978689 (s)	0.9514365 (s)	0.76999(s)	3.4203779(s)	1.3158315 (s)	0.969268 (s)
	10	0.775327 (s)	0.332367 (s)	0.2732847 (s)	0.7970195(s)	0.3335655 (s)	0.2830074 (s)	0.918433 (s)	0.390718(s)	0.319553 (s)
	20	0.579581(s)	0.272778(s)	0.2273388 (s)	0.6317591(s)	0.282981(s)	0.2361931s)	0.6429213(s)	0.2879445(s)	0.243859(s)
	30	0.518979(s)	0.272263 (s)	0.224307(s)	0.6254521(s)	0.282579 (s)	0.232619(s)	0.6110412(s)	0.286857 (s)	0.242472(s)
	40	0.447191(s)	0.265288(s)	0.2240797(s)	0.532283 (s)	0.2827365 (s)	0.236022(s)	0.5733182 (s)	0.2867265 (s)	0.242804(s)
	50	0.3673717 (s)	0.263392 (s)	0.221772(s)	0.3976215(s)	0.283255(s)	0.238583 (s)	0.5240913(s)	0.2877645 (s)	0.2477592 (s)
	60	0.384757 (s)	0.276195(s)	0.233776 (s)	0.4194903(s)	0.2839755(s)	0.239961(s)	0.552205(s)	0.296094 (s)	0.120813(s)
	70	0.335343(s)	0.276675 (s)	0.233781 (s)	0.4102117 (s)	0.288432(s)	0.2371473(s)	0.5383815 (s)	0.296454(s)	0.251826(s)
	80	0.302851(s)	0.278605 (s)	0.235151(s)	0.4029731(s)	0.307149 (s)	0.2594046(s)	0.4733786 (s)	0.3107325 (s)	0.263157 (s)

congestion and a long end-to-end delay. As the number of fog nodes increases the end-to-end delay decreases since fewer tuples are being sent to each fog node.

Effectiveness of Fog-Tree Hierarchy: From Tables 9.18, 9.19, 9.20, and 9.21we see that Tree 1 has the lowest end-to-end delay compared to Tree 2 and Tree 3. As we go up the fog-tree hierarchy the number of network hops increases and the processing power of fog nodes increases as well.

Effectiveness of Algorithms 8.1 and 8.2: From Tables 9.18, 9.19, 9.20, and 9.21 we see that

Table 9.20: Average End-to-End Delay-256 Cameras

		Tree 1			Tree 2			Tree 3		
	#CFNs	Original QG	Reconfigured QG	Adjusted QG	Original QG	Reconfigured QG	Adjusted QG	Original QG	Reconfigured QG	Adjusted QG
C_1	2	4.467373 (s)	2.828648 (s)	2.730123 (s)	4.60726 (s)	2.885391 (s)	2.81574 (s)	7.403225 (s)	3.38352 (s)	3.376091 (s)
	10	1.213663 (s)	0.878809 (s)	0.85887 (s)	1.245372 (s)	0.89564 (s)	0.87832 (s)	1.818515 (s)	0.989103 (s)	0.987628 (s)
	20	0.952543 (s)	0.744054 (s)	0.731465 (s)	0.976222 (s)	0.75522 (s)	0.746313 (s)	1.333287 (s)	0.804315 (s)	0.803386 (s)
	30	0.915198 (s)	0.688352 (s)	0.663751 (s)	0.956113 (s)	0.697051 (s)	0.688725 (s)	1.289708 (s)	0.778995 (s)	0.758211 (s)
	40	0.91073 (s)	0.681531 (s)	0.643987 (s)	0.692371 (s)	0.67256 (s)	0.658803 (s)	1.270152 (s)	0.773351 (s)	0.703721 (s)
	50	0.705957 (s)	0.624589 (s)	0.619046 (s)	0.725176 (s)	0.63747 (s)	0.634017 (s)	0.814809 (s)	0.71949 (s)	0.619149 (s)
	60	0.687382 (s)	0.617373 (s)	0.596329 (s)	0.707818 (s)	0.63256 (s)	0.58469 (s)	0.838829 (s)	0.638911 (s)	0.601621 (s)
	70	0.685301 (s)	0.617932 (s)	0.663201 (s)	0.800931 (s)	0.637024 (s)	0.566941 (s)	0.80112 (s)	0.659702 (s)	0.600972 (s)
	80	0.676606 (s)	0.617098 (s)	0.555303 (s)	0.699655 (s)	0.636855 (s)	0.604478 (s)	0.899865 (s)	0.654318 (s)	0.613669 (s)
C_2	2	9.628082 (s)	4.8262392 (s)	4.1678057(s)	9.929566(s)	4.9230542(s)	4.298508(s)	15.955052 (s)	5.7729618(s)	5.153052 (s)
	10	3.513686(s)	1.4994239(s)	1.311150942(s)	3.582073(s)	1.5281409(s)	1.340843(s)	4.817235(s)	1.68760753(s)	1.5077129 (s)
	20	2.9509206 (s)	1.269504(s)	1.1166544(s)	3.001965 (s)	1.288563 (s)	1.1393212(s)	4.669001 (s)	1.372322 (s)	1.2264406 (s)
	30	2.8704372 (s)	1.1744661(s)	1.01328227 (s)	2.958614(s)	1.1893081 (s)	1.051075(s)	4.575578(s)	1.3291212 (s)	1.157481 (s)
	40	2.860802 (s)	1.16282819 (s)	0.9831105(s)	3.288197(s)	1.1475218(s)	1.005726(s)	4.353815(s)	1.3194917(s)	1.074307(s)
	50	2.4194752 (s)	1.06573(s)	0.945036(s)	3.35889(s)	1.0876513(s)	0.967803(s)	4.00103(s)	1.227593(s)	0.9451928 (s)
	60	2.379446(s)	1.0533611(s)	0.910355(s)	3.321435(s)	1.0792732 (s)	0.892587(s)	3.963044(s)	1.0901099 (s)	0.918434 (s)
	70	2.3749607(s)	1.0543155(s)	1.0124426 (s)	3.522164(s)	1.08689(s)	0.865492(s)	3.79196(s)	1.12558(s)	0.9174438(s)
	80	2.356228(s)	1.0528926 (s)	0.84772555 (s)	3.303896(s)	1.086602 (s)	0.9227961(s)	3.735389 (s)	1.1163977(s)	0.936827 (s)
C_3	2	10.721695 (s)	5.3744312 (s)	4.641209(s)	11.057424(s)	5.48224(s)	4.786758(s)	17.76774 (s)	6.428688 (s)	5.7393547 (s)
	10	3.912791 (s)	1.669737(s)	1.460079(s)	3.988898(s)	1.701716(s)	1.493144(s)	5.364436(s)	1.8792957(s)	1.6789676 (s)
	20	3.286103 (s)	1.4137026(s)	1.2434905(s)	3.34298 (s)	1.434918 (s)	1.2687321(s)	5.199888 (s)	1.5281985 (s)	1.365752 (s)
	30	3.196475 (s)	1.307868(s)	1.1283767 (s)	3.2946712 (s)	1.3243969 (s)	1.170835(s)	5.09529(s)	1.480005 (s)	1.288958 (s)
	40	3.185752 (s)	1.2949089 (s)	1.094777(s)	3.6616904 (s)	1.277864(s)	1.119961 (s)	4.848364(s)	1.469366(s)	1.196327(s)
	50	2.694296 (s)	1.1867191(s)	1.0523782(s)	3.740424 (s)	1.211193 (s)	1.077829 (s)	4.45554(s)	1.367031(s)	1.052553 (s)
	60	2.649716 (s)	1.1730087(s)	1.013759(s)	3.698732(s)	1.201864 (s)	0.993973 (s)	4.413189(s)	1.2139309 (s)	1.022757 (s)
	70	2.644722(s)	1.1740708(s)	1.1274417 (s)	3.92223(s)	1.210345(s)	0.963799(s)	4.22268(s)	1.253433(s)	1.021652(s)
	80	2.623854(s)	1.1724862 (s)	0.9440151 (s)	3.679172(s)	1.2100245 (s)	1.027612(s)	4.159676 (s)	1.243204(s)	1.043237 (s)

Table 9.21: Average End-to-End Delay-512 Cameras

		Tree 1			Tree 2			Tree 3		
	#CFNs	Original QG	Reconfigured QG	Adjusted QG	Original QG	Reconfigured QG	Adjusted QG	Original QG	Reconfigured QG	Adjusted QG
C_1	2	23.75610651(s)	7.6373496 (s)	3.52185867(s)	27.957752 (s)	9.03127383 (s)	5.9975262(s)	47.938705 (s)	10.6919232 (s)	8.67655387(s)
	10	9.91053881(s)	2.3727843(s)	1.1079423(s)	10.4759344 (s)	2.8033532(s)	1.8708216(s)	15.547387(s)	3.12556548 (s)	2.53820396 (s)
	20	9.63888441(s)	2.0089458 (s)	0.94358985 (s)	9.0763544 (s)	2.3638386 (s)	1.58964669(s)	13.7330646(s)	2.5416354 (s)	2.06470202 (s)
	30	8.45701426(s)	1.8585504 (s)	0.85623879(s)	9.9717876(s)	2.18176963(s)	1.46698425 (s)	13.4803064(s)	2.4616242 (s)	1.94860227(s)
	40	8.4352551(s)	1.8401337(s)	0.83074323(s)	8.6003292(s)	2.1051128 (s)	1.40325039(s)	12.3668816(s)	2.44378916(s)	1.80856297(s)
	50	8.43801059(s)	1.6863903 (s)	0.79856934 (s)	8.7709152(s)	1.9952811(s)	1.35045621(s)	12.7258922(s)	2.2735884(s)	1.59121293(s)
	60	7.34755034(s)	1.6669071(s)	0.76926441(s)	8.6806536(s)	1.9799128(s)	1.2453897(s)	11.8652082(s)	2.01895876 (s)	1.54616597(s)
	70	7.33741587(s)	1.6684164 (s)	0.75552929(s)	9.1648412(s)	1.99388512 (s)	1.20758433(s)	10.646496 (s)	2.08465832(s)	1.54449804 (s)
	80	7.29507122(s)	1.6661646 (s)	0.71634087 (s)	9.638206(s)	1.99335615 (s)	1.28753814 (s)	10.219217 (s)	2.3555448 (s)	1.57712933(s)
C_2	2	58.048045(s)	15.205962(s)	7.012026(s)	68.314767 (s)	17.9812661 (s)	11.941074 (s)	117.138225(s)	21.287619 (s)	17.27501(s)
	10	24.216408(s)	4.7242134(s)	2.2059131(s)	25.59794(s)	5.581476(s)	3.7248058 (s)	37.9900401 (s)	6.2230008 (s)	5.053564 (s)
	20	23.552613(s)	3.999811 (s)	1.878687(s)	22.17807(s)	4.7064026 (s)	3.1649865 (s)	33.55674 (s)	5.060395(s)	4.110821 (s)
	30	20.664706(s)	3.700373 (s)	1.70477142(s)	24.36606(s)	4.343903(s)	2.920765(s)	32.93912(s)	4.9010937 (s)	3.879667(s)
	40	20.61154(s)	3.663706(s)	1.6540097(s)	21.01489(s)	4.1912795 (s)	2.793871(s)	30.2184751 (s)	4.8655841 (s)	3.600848(s)
	50	20.61827(s)	3.35760 (s)	1.589951(s)	21.431731(s)	3.972604(s)	2.688758 (s)	31.0957167 (s)	4.526714(s)	3.1681048 (s)
	60	18.95373(s)	3.3188114(s)	1.5316054(s)	21.21117(s)	3.9420063(s)	2.4795708(s)	28.99263 (s)	4.019746 (s)	3.0784164(s)
	70	18.92897(s)	3.321816 (s)	1.5042587(s)	22.394289 (s)	3.8793252 (s)	2.40429(s)	26.0147129(s)	4.150554(s)	3.075095 (s)
	80	18.8255(s)	3.31733 (s)	1.4262346(s)	23.55095(s)	3.70384517 (s)	2.563488(s)	25.33265 (s)	3.785101(s)	3.1400644(s)
C_3	2	64.141487(s)	16.802169(s)	7.748089(s)	75.4859304 (s)	19.868802 (s)	13.1945576 (s)	129.434503 (s)	23.522231 (s)	19.088418 (s)
	10	26.758454(s)	5.2201254(s)	2.437473(s)	28.285022(s)	6.16737704(s)	4.11580752 (s)	41.9779449 (s)	6.876244 (s)	5.584048 (s)
	20	26.024987(s)	4.4196807 (s)	2.075897 (s)	24.506156 (s)	5.200444 (s)	3.4972227 (s)	37.079274 (s)	5.591597 (s)	4.542344 (s)
	30	22.83393(s)	4.0888108 (s)	1.883725(s)	26.9236 (s)	4.799893(s)	3.227365 (s)	36.396827 (s)	5.41557324 (s)	4.2869249(s)
	40	22.775188(s)	4.048294(s)	1.8276351(s)	23.22088 (s)	4.631248 (s)	3.0871508 (s)	33.390583 (s)	5.3763361 (s)	3.97888(s)
	50	22.782628(s)	3.710058 (s)	1.75685(s)	23.681471 (s)	4.3896184(s)	2.9710036 (s)	34.359908 (s)	5.0018944(s)	3.500663 (s)
	60	19.838385(s)	3.667195(s)	1.692381(s)	23.437764 (s)	4.355808(s)	2.739834(s)	32.0360621 (s)	4.4417092 (s)	3.4015651(s)
	70	19.811028(s)	3.670516 (s)	1.662164(s)	24.7450712 (s)	4.2865472 (s)	2.65668(s)	28.745532 (s)	4.5862404(s)	3.397895 (s)
	80	19.696692(s)	3.665562 (s)	1.575941(s)	26.023156 (s)	4.1153835 (s)	2.832583(s)	27.991889 (s)	4.1824321(s)	3.469684(s)

the original query graph results in the highest end-to-end delay compared to the reconfigured and adjusted query graphs. A fog node is assigned only one node of the original graph, while Algorithm 8.1 and Algorithm 8.2 are used to map one or more nodes (representing multiple data operations) to a single fog node and this reduced the network latency due to the amount of data that needs to be transferred among fog nodes.

Comparison of Cost Functions C_1, C_2, and C_3: From Tables 9.18, 9.19, 9.20, and 9.21 we see that cost function C_1 has lower end-to-end delay compared to cost functions C_2 and C_3.

Table 9.22: Average End-to-End Delay-640 Cameras

		Tree 1			Tree 2			Tree 3		
	#CFNs	Original QG	Reconfigured QG	Adjusted QG	Original QG	Reconfigured QG	Adjusted QG	Original QG	Reconfigured QG	Adjusted QG
C_1	2	30.882934(s)	9.928554(s)	4.5784162(s)	36.3456 (s)	11.740655 (s)	7.796706(s)	62.320315 (s)	13.899001 (s)	11.279501(s)
	10	12.883004(s)	3.084619(s)	1.440324(s)	13.618714(s)	3.644358(s)	2.432068(s)	20.211601(s)	4.06324(s)	3.299664 (s)
	20	12.53097(s)	2.611654 (s)	1.22666(s)	11.79926(s)	3.072998 (s)	2.06654(s)	17.85298(s)	3.304126(s)	2.684112(s)
	30	10.994118(s)	2.416115(s)	1.113117(s)	12.963323(s)	2.836305(s)	1.907079 (s)	17.52439(s)	3.200111 (s)	2.533182(s)
	40	10.965831(s)	2.392173(s)	1.079966(s)	11.180427(s)	2.736646(s)	1.824225(s)	16.076946(s)	3.176925(s)	2.351131(s)
	50	10.969417(s)	2.192303 (s)	1.0381401 (s)	11.402189(s)	2.59365(s)	1.755593(s)	16.543659(s)	2.955664(s)	2.068576(s)
	60	9.55154(s)	2.166979(s)	1.000047(s)	11.284849(s)	2.573886(s)	1.619006(s)	15.424776(s)	2.624648 (s)	2.010015(s)
	70	9.53831(s)	2.168941(s)	0.982187(s)	11.914293(s)	2.5920506 (s)	1.569859(s)	13.84044(s)	2.71006(s)	2.0078474 (s)
	80	9.483592(s)	2.1660139 (s)	0.9312431(s)	12.529667(s)	2.591362 (s)	1.673799 (s)	13.284982 (s)	3.062208 (s)	2.050268(s)
C_2	2	78.364865(s)	20.528048(s)	9.4662281(s)	92.224945 (s)	24.2747035 (s)	16.1204499 (s)	158.136603(s)	28.738265 (s)	23.32125(s)
	10	32.692141(s)	6.377688(s)	2.9779826(s)	34.557219(s)	7.53499287(s)	5.02848783 (s)	51.2865541 (s)	8.401108(s)	6.822114 (s)
	20	31.7960275(s)	5.399743 (s)	2.536227(s)	29.94039(s)	6.3536435 (s)	4.272731(s)	45.301599(s)	6.831533(s)	5.549605 (s)
	30	27.89735(s)	4.995503 (s)	2.301441(s)	32.8941(s)	5.864269(s)	3.943032(s)	44.46781(s)	6.6164764 (s)	5.23755(s)
	40	27.825579(s)	4.946(s)	2.232913(s)	28.370101(s)	5.658273 (s)	3.771725(s)	40.7949413 (s)	6.568538 (s)	4.861144(s)
	50	27.83466(s)	4.5327(s)	2.146433(s)	28.9326(s)	5.36301(s)	3.629823(s)	41.979217545(s)	6.111063(s)	4.2769414 (s)
	60	25.58755(s)	4.480395(s)	2.067667(s)	28.635079(s)	5.321085(s)	3.34742(s)	39.14005 (s)	5.4266571 (s)	4.155864(s)
	70	25.5541095(s)	4.484451 (s)	2.0307375(s)	30.232901 (s)	5.23089(s)	3.24579(s)	35.119862(s)	5.603248(s)	4.151375 (s)
	80	25.414425(s)	4.4783955 (s)	1.92541671(s)	31.793725(s)	5.000197 (s)	3.460708(s)	34.199075(s)	5.109886(s)	4.239084(s)
C_3	2	89.15666(s)	23.355011(s)	10.769843(s)	104.925436 (s)	27.617638(s)	18.34034(s)	179.91357(s)	32.69589 (s)	26.5329 (s)
	10	37.194206(s)	7.255974(s)	3.3880874(s)	39.31618(s)	8.572654(s)	5.72097(s)	58.34934(s)	9.557976 (s)	7.7618276 (s)
	20	36.174731(s)	6.1433561(s)	2.885497(s)	34.063556 (s)	7.22864 (s)	4.861139 (s)	51.54019(s)	7.772319 (s)	6.313858(s)
	30	31.73916(s)	5.683447(s)	2.618378(s)	37.424118 (s)	6.6718514(s)	4.486037(s)	50.59158(s)	7.5276468 (s)	5.958825(s)
	40	31.65751(s)	5.627128(s)	2.5404127(s)	32.2770 (s)	6.437434(s)	4.291139(s)	46.412906 (s)	7.4731071 (s)	5.530585(s)
	50	31.667853(s)	5.15698 (s)	2.4420247(s)	32.917269(s)	6.10156(s)	4.129695(s)	47.760272(s)	6.95263(s)	4.865929 (s)
	60	27.575355(s)	5.097405(s)	2.35246(s)	32.578496 (s)	6.054574(s)	3.8084017(s)	44.53012 (s)	6.173957 (s)	4.728175(s)
	70	27.537321(s)	5.102017(s)	2.310405(s)	34.39564 (s)	5.9583 (s)	3.692752(s)	39.956299 (s)	6.374885(s)	4.72307 (s)
	80	27.3784(s)	5.0951311 (s)	2.1905703(s)	36.172186(s)	5.7203831 (s)	3.937221(s)	39.160647(s)	5.81358(s)	4.822145(s)

This is expected since the cost function C_1 favours the mapping of the query vertices as close as possible to the data sources.

9.3.3 Average Response Time

Tables 9.23, 9.24, 9.25, and 9.26 show the average response time of the processing of the original query graph, reconfigured query graph, and adjusted query graph as the number of fog nodes and cameras vary as well as for the different types of trees.

Effectiveness of Collector Fog Nodes: The results in Tables 9.23, 9.24, 9.25, 9.26, and 9.27 show that when the number of CFNs increases the average response time decreases significantly for both reconfigured and adjusted query graphs.

Effectiveness Fog-Tree Hierarchy: Tree 3, representing the shortest tree, has the lowest response time as the result of providing more CFNs closer to data sources than the other two trees.

Effectiveness of Algorithms 8.1 and 8.2: From Tables 9.23, 9.24, 9.25, 9.26, and 9.27 we see that the original query graph results in less response time compared to the reconfigured and the adjusted query graph. A fog node is assigned only one node (representing an data operation) of the original graph, while Algorithm 8.1 and Algorithm 8.2 are used to map one or more nodes to a single fog node. The response time of the reconfigured and adjusted query graphs is significantly higher for a small number of CFNs. However, when the number of CFNs increases the differences decrease.

Effectiveness of Fog-Tree Hierarchy: By comparing the results in Tables 9.23, 9.24, 9.25, 9.26, and 9.27 we can conclude that the fog-tree hierarchy has a good tolerance for processing the huge volume of data. The results in tables 9.23, 9.24, 9.25, 9.26, and 9.27 show that the increases of the number of cameras were not significantly change the average response time when the number of level 0 fog nodes are sufficient. As the results show, Tree 3 (which is a

Table 9.23: Average Response Time-64 Cameras

		Tree 1			Tree 2			Tree 3		
	#CFNs	Original QG	Reconfigured QG	Adjusted QG	Original QG	Reconfigured QG	Adjusted QG	Original QG	Reconfigured QG	Adjusted QG
C_1	2	930.79295 (s)	8044.5122 (s)	8045.54471 (s)	892.20654 (s)	7824.5338 (s)	7827.4038 (s)	728.58071 (s)	6896.70831 (s)	6898.7637 (s)
	10	17.13022 (s)	1289.25459 (s)	1291.21105 (s)	16.63794 (s)	1241.86029 (s)	1243.2739 (s)	14.8779 (s)	1038.22601 (s)	1049.177 (s)
	20	10.89046 (s)	480.66343 (s)	480.9673 (s)	10.67025 (s)	454.9124 (s)	463.10963 (s)	9.45477 (s)	336.18338 (s)	337.91409 (s)
	30	10.7105 (s)	40.35513 (s)	50.44917 (s)	10.05913 (s)	27.96921 (s)	30.474091 (s)	7.749013 (s)	23.986742 (s)	24.1921 (s)
	40	10.14902 (s)	32.66901 (s)	36.548 (s)	9.50008 (s)	27.09831 (s)	27.917513 (s)	7.54336 (s)	23.91866 (s)	24.18034 (s)
	50	9.12656 (s)	29.06189 (s)	29.46036 (s)	8.52551 (s)	27.08961 (s)	27.4393 (s)	7.51403 (s)	24.14516 (s)	24.2451 (s)
	60	9.11215 (s)	28.09244 (s)	29.20691 (s)	8.13732 (s)	26.22004 (s)	27.221321 (s)	7.21793 (s)	23.13219 (s)	24.108 (s)
	70	9.30273 (s)	27.88073 (s)	28.48791 (s)	8.53974 (s)	26.22652 (s)	27.4293 (s)	7.49309 (s)	19.12437 (s)	19.96217 (s)
	80	9.52067 (s)	17.07887 (s)	17.27289 (s)	8.55516 (s)	15.09371 (s)	15.92516 (s)	7.45096 (s)	14.03582 (s)	14.0358 (s)
C_2	2	911.45043 (s)	719.4912 (s)	722.229(s)	874.846(s)	709.603(s)	711.4941(s)	714.019 (s)	625.783(s)	622.266 (s)
	10	17.37958 (s)	109.9487(s)	112.239(s)	17.49181 (s)	113.6111(s)	114.722347(s)	15.79403(s)	91.222(s)	93.606(s)
	20	11.876305 (s)	40.677 (s)	40.543(s)	11.626(s)	38.35(s)	39.633 (s)	10.44(s)	28.636(s)	29.774(s)
	30	11.6367(s)	5.9926 (s)	5.664(s)	11.02808 (s)	3.745 (s)	3.645(s)	8.76758(s)	3.3878(s)	3.1908(s)
	40	11.134782(s)	4.156(s)	4.0794(s)	10.45571(s)	3.666(s)	3.64166(s)	8.55912(s)	3.382(s)	3.07702(s)
	50	9.632837(s)	3.919(s)	3.773(s)	9.487(s)	3.665(s)	3.6331(s)	8.5218 (s)	3.401861(s)	3.94215(s)
	60	9.61776(s)	3.7559(s)	3.72796(s)	9.112(s)	3.589 (s)	3.58051(s)	8.2373(s)	3.3161(s)	3.8464(s)
	70	9.49596(s)	3.738(s)	3.73709(s)	9.503(s)	3.6005(s)	3.30785(s)	8.5036(s)	2.95916(s)	3.8367(s)
	80	9.3143 (s)	2.7623(s)	2.724(s)	9.51213(s)	2.6203(s)	2.60479(s)	8.512507(s)	2.5062(s)	3.488(s)
C_3	2	820.635(s)	645.118 (s)	633.085 (s)	786.699 (s)	627.538 (s)	623.685 (s)	641.511 (s)	555.092 (s)	545.515 (s)
	10	15.728 (s)	104.315 (s)	98.394 (s)	15.832 (s)	100.4807 (s)	100.571(s)	14.321 (s)	84.23(s)	82.071 (s)
	20	10.747 (s)	39.522 (s)	35.55 (s)	10.53 (s)	37.473 (s)	32.125 (s)	9.464 (s)	27.993 (s)	25.115 (s)
	30	10.532 (s)	5.317(s)	4.104 (s)	9.99 (s)	3.332 (s)	3.212 (s)	7.954 (s)	3.016 (s)	2.816 (s)
	40	10.061 (s)	3.696 (s)	3.591 (s)	9.47 (s)	3.262 (s)	3.21 (s)	7.761 (s)	3.011 (s)	1.716 (s)
	50	10.698(s)	3.482 (s)	3.322 (s)	8.571 (s)	3.262 (s)	3.203 (s)	7.72 (s)	3.029 (s)	2.598 (s)
	60	10.687 (s)	3.3407 (s)	3.285 (s)	8.236 (s)	3.194 (s)	3.157 (s)	7.468 (s)	2.955 (s)	2.515 (s)
	70	10.57 (s)	3.325 (s)	3.293 (s)	8.588 (s)	3.207 (s)	2.921 (s)	7.706 (s)	2.64 (s)	2.508 (s)
	80	10.402 (s)	2.462 (s)	2.406 (s)	8.594 (s)	2.341 (s)	2.305 (s)	7.704 (s)	2.241 (s)	2.204 (s)

Table 9.24: Average Response Time-128 Cameras

		Tree 1			Tree 2			Tree 3		
	#CFNs	Original QG	Reconfigured QG	Adjusted QG	Original QG	Reconfigured QG	Adjusted QG	Original QG	Reconfigured QG	Adjusted QG
C_1	2	1144.9398 (s)	10861.5167 (s)	11181.803 (s)	1097.4772 (s)	10567.0132 (s)	10719.5553 (s)	896.1899 (s)	9313.4012 (s)	9448.4145 (s)
	10	22.15807 (s)	1743.21857 (s)	1769.34121 (s)	20.55183 (s)	1678.46337 (s)	1721.44169 (s)	18.3842 (s)	1404.3164 (s)	1432.4208 (s)
	20	14.4869 (s)	648.97885 (s)	659.00487 (s)	13.21462 (s)	614.4476 (s)	622.2396 (s)	11.761189 (s)	456.2659 (s)	461.1615 (s)
	30	14.26628 (s)	57.5635 (s)	71.19554 (s)	13.06291 (s)	37.754386 (s)	43.2239 (s)	10.6224 (s)	32.46423 (s)	33.219517 (s)
	40	13.57705 (s)	44.18785 (s)	51.15036 (s)	12.7771 (s)	38.205491 (s)	38.7685 (s)	10.3702 (s)	32.37215 (s)	33.203773 (s)
	50	12.3211 (s)	40.85859 (s)	42.8935 (s)	12.58089 (s)	37.11883 (s)	37.31152 (s)	10.3342 (s)	32.8138 (s)	33.15958 (s)
	60	12.3033 (s)	39.56812 (s)	39.9088 (s)	12.10375 (s)	36.69731 (s)	36.8195 (s)	9.9648 (s)	31.77039 (s)	32.52193 (s)
	70	12.53826 (s)	39.2784 (s)	39.5378 (s)	11.5983 (s)	26.72128 (s)	35.3234 (s)	10.3086 (s)	26.2726 (s)	27.02071 (s)
	80	12.80714 (s)	24.40273 (s)	24.4748 (s)	11.53804 (s)	21.7557 (s)	23.5735 (s)	10.2584 (s)	19.2691 (s)	19.80142 (s)
C_2	2	1126.261 (s)	1013.923 (s)	1027.38 (s)	1079.711(s)	962.136(s)	978.1406 (s)	882.669(s)	853.7225(s)	895.93 (s)
	10	24.22 (s)	162.22199(s)	163.1726(s)	22.651(s)	153.0311(s)	158.804 (s)	20.564(s)	121.961(s)	124.471(s)
	20	16.63 (s)	56.867 (s)	57.7556(s)	15.3995 (s)	57.242(s)	58.964 (s)	13.97694 (s)	40.554 (s)	41.729(s)
	30	16.709(s)	8.20357 (s)	8.7845(s)	15.2486(s)	6.3774 (s)	6.594(s)	12.84889 (s)	5.5937(s)	5.4866(s)
	40	15.713(s)	7.9166(s)	7.9604(s)	14.8977 (s)	5.9759 (s)	6.7453 (s)	12.5897(s)	5.4473921(s)	5.25881 (s)
	50	14.406 (s)	6.0316(s)	5.9697132(s)	14.669(s)	5.997(s)	5.855(s)	12.537(s)	5.6272 (s)	5.4857 (s)
	60	14.591(s)	6.47854(s)	5.899(s)	14.2432(s)	5.9961 (s)	5.811(s)	12.184978 (s)	5.4018(s)	5.0957(s)
	70	14.582(s)	6.10435 (s)	5.936(s)	13.744(s)	5.5074(s)	6.0119 (s)	12.52154 (s)	5.009512(s)	5.89805(s)
	80	13.8217(s)	4.6548(s)	4.583(s)	13.7288(s)	4.67487(s)	4.423(s)	12.44794(s)	4.24947(s)	4.1659487(s)
C_3	2	1010.481 (s)	897.451 (s)	817.236 (s)	968.743 (s)	855.423 (s)	800.283 (s)	792.294 (s)	759.128 (s)	671.462 (s)
	10	21.872 (s)	150.486(s)	143.821 (s)	20.47 (s)	142.983 (s)	139.975(s)	18.621(s)	114.71(s)	109.733 (s)
	20	15.027 (s)	55.0477 (s)	50.933 (s)	13.933 (s)	54.494 (s)	52.0001 (s)	12.659 (s)	38.772 (s)	36.816 (s)
	30	15.086 (s)	7.346 (s)	6.905 (s)	13.797 (s)	5.725 (s)	4.977 (s)	11.642 (s)	5.029 (s)	4.884 (s)
	40	14.179 (s)	7.09 (s)	5.298 (s)	13.464 (s)	5.368 (s)	5.111 (s)	11.402 (s)	4.899 (s)	4.683(s)
	50	12.991 (s)	5.414 (s)	5.3057 (s)	13.233 (s)	5.388 (s)	5.208 (s)	11.346 (s)	5.059 (s)	4.884 (s)
	60	13.16 (s)	5.814 (s)	4.365 (s)	12.855 (s)	5.386 (s)	5.169 (s)	11.035 (s)	4.861 (s)	4.514 (s)
	70	13.143 (s)	5.481 (s)	4.397 (s)	12.406 (s)	4.953 (s)	3.465 (s)	11.334 (s)	4.512 (s)	4.367 (s)
	80	12.454 (s)	4.193 (s)	4.087 (s)	12.39 (s)	4.217 (s)	3.951 (s)	11.255 (s)	3.839 (s)	3.725 (s)

shortest tree among Tree 1 and Tree 2) has the lowest response time. The explanation is that Tree 3 provides more processing power closer to the data sources and this results in lower the execution time.

Comparison of Cost Functions: From Tables 9.23, 9.24, 9.25, 9.26, and 9.27 we see that cost function C_3 has a lower response time compared to cost functions C_1 and C_2. The reason is that the cost function C_3 considers both execution time and network latency at the same time for mapping vertices of a query graph.

Table 9.25: Average Response Time-256 Cameras

		Tree 1			Tree 2			Tree 3		
	#CFNs	Original QG	Reconfigured QG	Adjusted QG	Original QG	Reconfigured QG	Adjusted QG	Original QG	Reconfigured QG	Adjusted QG
C_1	2	5523.6423 (s)	33980.0208 (s)	33982.8973 (s)	5369.536 (s)	33110.3191 (s)	33110.374 (s)	4664.8675 (s)	29384.41652 (s)	29391.4591 (s)
	10	644.963 (s)	6424.7018 (s)	6444.69287 (s)	613.5753 (s)	6236.33664 (s)	6240.10932 (s)	480.250515 (s)	5492.0131 (s)	5492.31628 (s)
	20	167.39454 (s)	3720.6634 (s)	3723.2465 (s)	163.2872 (s)	3613.0072 (s)	3614.91233 (s)	65.98828 (s)	3144.89835 (s)	3149.4243 (s)
	30	99.432198 (s)	2727.7963 (s)	2731.051751 (s)	99.1713 (s)	2726.6241 (s)	2730.8327 (s)	19.671708 (s)	2046.2209 (s)	2067.484211 (s)
	40	91.12273 (s)	1820.65353 (s)	1822.3809 (s)	90.890371 (s)	1817.6935 (s)	1818.0368 (s)	17.752152 (s)	1542.07635 (s)	1552.964721 (s)
	50	18.702957 (s)	1018.45758 (s)	1021.24504 (s)	18.25817 (s)	979.91247 (s)	980.255 (s)	16.457809 (s)	802.9854 (s)	805.73114 (s)
	60	17.784382 (s)	1009.97337 (s)	1012.42932 (s)	18.12181 (s)	966.54656 (s)	966.59769 (s)	15.770829 (s)	791.370911 (s)	793.6296 (s)
	70	17.127301 (s)	492.5559 (s)	493.435201 (s)	15.692931 (s)	465.169024 (s)	466.907941 (s)	14.03512 (s)	338.6897 (s)	341.8939 (s)
	80	16.7756 (s)	478.609 (s)	480.9483 (s)	15.573655 (s)	453.43885 (s)	453.761478 (s)	14.013865 (s)	337.586318 (s)	338.2466 (s)
C_2	2	5038.9754 (s)	2897.128 (s)	2903.589(s)	4898.72 (s)	2492.823 (s)	2503.907 (s)	4260.065 (s)	2393.329 (s)	2402.927 (s)
	10	590.125 (s)	548.257 (s)	582.029 (s)	561.278(s)	461.5496 (s)	470.505 (s)	440.787 (s)	469.109(s)	467.884(s)
	20	154.621(s)	317.849 (s)	378.169(s)	150.905(s)	308.7815(s)	320.122 (s)	63.6119 (s)	269.0132 (s)	280.163(s)
	30	92.642(s)	202.2987(s)	210.478(s)	92.4616 (s)	202.294(s)	208.251 (s)	21.317 (s)	175.4472(s)	180.622(s)
	40	85.0668(s)	135.431 (s)	136.1332 (s)	85.48(s)	155.819 (s)	156.259 (s)	20.499(s)	132.522 (s)	136.564(s)
	50	23.296 (s)	87.5015(s)	87.1402(s)	20.4622(s)	84.444 (s)	83.8757 (s)	19.945(s)	60.403 (s)	59.7461 (s)
	60	22.4573(s)	86.868(s)	86.348 (s)	20.3146(s)	83.31 (s)	72.7871(s)	19.818(s)	59.4268 (s)	57.569 (s)
	70	20.725(s)	43.0152(s)	37.6054(s)	20.466 (s)	35.361 (s)	35.189 (s)	19.225 (s)	26.063 (s)	24.836 (s)
	80	20.409(s)	36.316(s)	35.23(s)	20.228(s)	32.493 (s)	32.965(s)	19.623(s)	20.5713 (s)	21.7167 (s)
C_3	2	4481.252(s)	2553.658 (s)	2528.153 (s)	4356.649(s)	2197.465 (s)	2187.851 (s)	3790.311 (s)	2110.003 (s)	2099.864(s)
	10	525.345(s)	483.394 (s)	421.301 (s)	499.718 (s)	407.006 (s)	385.043 (s)	392.893 (s)	413.705 (s)	391.52(s)
	20	138.104 (s)	280.338 (s)	243.21 (s)	134.813 (s)	272.353 (s)	236.188 (s)	57.574(s)	237.335 (s)	227.511 (s)
	30	82.993 (s)	178.509(s)	166.6 (s)	82.853 (s)	178.509 (s)	155.93 (s)	19.9772 (s)	154.888 (s)	149.292 (s)
	40	76.257(s)	119.592 (s)	116.509 (s)	76.721 (s)	137.552 (s)	119.245 (s)	19.2 (s)	117.066 (s)	102.0613 (s)
	50	21.251 (s)	77.341(s)	67.598(s)	18.943 (s)	74.653 (s)	64.7528 (s)	18.628 (s)	53.514 (s)	52.407 (s)
	60	20.496(s)	76.781 (s)	66.89 (s)	18.803 (s)	73.651 (s)	63.78 (s)	18.507 (s)	52.611 (s)	50.499 (s)
	70	18.956 (s)	38.144 (s)	33.086 (s)	18.984 (s)	31.408 (s)	30.941 (s)	17.941(s)	23.225 (s)	21.911 (s)
	80	18.67 (s)	32.241 (s)	30.973 (s)	18.723 (s)	30.643 (s)	29.0126 (s)	17.393 (s)	20.146 (s)	18.318 (s)

Table 9.26: Average Response Time-512 Cameras

		Tree 1			Tree 2			Tree 3		
	#CFNs	Original QG	Reconfigured QG	Adjusted QG	Original QG	Reconfigured QG	Adjusted QG	Original QG	Reconfigured QG	Adjusted QG
C_1	2	15422.254(s)	95483.546(s)	95147.989(s)	11777.1588(s)	73838.604(s)	75160.097(s)	9875.1877(s)	64061.343(s)	63780.816 (s)
	10	1805.9814 (s)	18053.315 (s)	18043.843 (s)	1351.4786 (s)	13907.836 (s)	14164.925 (s)	1025.0385 (s)	11973.5578 (s)	11918.726 (s)
	20	474.012 (s)	10454.981(s)	10423.985 (s)	364.5373 (s)	8166.053(s)	8205.746 (s)	150.155(s)	6856.66 (s)	7149.4343 (s)
	30	283.319(s)	7665.0319(s)	7782.462(s)	225.0757 (s)	6162.7767 (s)	6198.893 (s)	52.2663 (s)	4461.5251 (s)	4486.744 (s)
	40	260.1267 (s)	5115.961(s)	5101.694(s)	206.1339 (s)	4126.7427 (s)	4308.589 (s)	47.1439(s)	3362.4843 (s)	3370.2149 (s)
	50	58.6496(s)	2861.797(s)	2858.551(s)	47.1681 (s)	2224.9495 (s)	2225.0901 (s)	45.732(s)	1751.2134(s)	1748.6842 (s)
	60	55.0481 (s)	2837.957(s)	2833.901(s)	46.817 (s)	2194.6046 (s)	2194.0948 (s)	43.3717(s)	1725.8147 (s)	1722.4169 (s)
	70	53.2105 (s)	1384.0141(s)	1380.517(s)	41.7783 (s)	1051.8362 (s)	1059.8016 (s)	38.5701 (s)	738.99 (s)	742.1503 (s)
	80	52.2112 (s)	1344.823 (s)	1345.816 (s)	42.2122 (s)	1034.381 (s)	1039.017(s)	37.889 (s)	736.867 (s)	734.24 (s)
C_2	2	12344.586 (s)	7027.801(s)	7107.523(s)	9443.059(s)	4244.723(s)	4365.116(s)	7958.3492(s)	3688.186(s)	3728.175(s)
	10	1457.308 (s)	1370.481 (s)	1452.566(s)	1095.587 (s)	916.147 (s)	969.7241(s)	843.462(s)	805.768(s)	805.344 (s)
	20	394.077 (s)	662.2748(s)	693.188(s)	305.802(s)	472.0793(s)	467.72498(s)	141.881(s)	397.458(s)	383.923(s)
	30	239.964 (s)	486.2876 (s)	483.218(s)	195.796(s)	356.793(s)	371.5547(s)	63.886(s)	259.294(s)	256.919 (s)
	40	221.4365 (s)	325.717(s)	339.642(s)	178.626 (s)	240.33(s)	240.41387(s)	57.966(s)	235.6995(s)	227.79 (s)
	50	60.6812(s)	183.472(s)	184.987(s)	54.0687(s)	131.239 (s)	123.298(s)	57.4311(s)	124.678(s)	123.084(s)
	60	57.01273 (s)	147.582(s)	148.565(s)	53.6401 (s)	129.469(s)	119.792(s)	54.131(s)	76.37(s)	69.204 (s)
	70	55.37397(s)	67.4748(s)	62.47025(s)	53.41628(s)	56.983 (s)	53.7456(s)	48.2847(s)	46.337(s)	42.848(s)
	80	54.6635(s)	60.0013(s)	52.991 (s)	53.294(s)	56.806(s)	52.832(s)	47.41(s)	33.3(s)	31.277(s)
C_3	2	10889.285 (s)	5267.977 (s)	5116.416 (s)	8335.173 (s)	3711.346(s)	3522.437 (s)	7037.99 (s)	3226.054 (s)	3145.416(s)
	10	1289.395 (s)	1198.0211 (s)	1085.001 (s)	971.009 (s)	801.422 (s)	753.898 (s)	751.649(s)	705.397 (s)	620.555 (s)
	20	352.478 (s)	579.332 (s)	427.455 (s)	274.395 (s)	413.386 (s)	405.717(s)	132.983 (s)	348.297 (s)	333.384 (s)
	30	216.061 (s)	425.562(s)	418.779 (s)	178.141 (s)	312.82 (s)	305.075 (s)	63.662 (s)	227.593(s)	223.368(s)
	40	199.714(s)	285.318 (s)	207.878 (s)	162.086 (s)	210.866 (s)	191.503 (s)	57.838 (s)	206.9783 (s)	198.082(s)
	50	58.08 (s)	161.016 (s)	143.221 (s)	50.674 (s)	115.539 (s)	107.395 (s)	57.5629(s)	109.937 (s)	107.324 (s)
	60	53.371 (s)	129.662(s)	111.679(s)	50.247 (s)	113.986 (s)	104.309 (s)	54.185(s)	67.629 (s)	60.653(s)
	70	51.922 (s)	59.699(s)	54.447(s)	47.672(s)	56.778 (s)	53.168 (s)	48.367(s)	41.431(s)	37.833 (s)
	80	51.272 (s)	54.538(s)	52.281 (s)	47.824 (s)	55.733 (s)	51.55 (s)	47.443(s)	32.58(s)	30.428 (s)

9.3.4 Average Distortion Due to Buffering

Tables 9.28, 9.29, 9.30, 9.30, 9.31, and 9.32 show the average distortion due to buffering for processing of the original query graph, reconfigured query graph, and adjusted query graph as the number of fog nodes and cameras vary.

The observations made from Tables 9.28, 9.29, 9.30, 9.30, 9.31, and 9.32 are described in the rest of this section.

Effectiveness of Collector Fog Nodes: The results in Tables 9.28, 9.29, 9.30, 9.30, 9.31,

Table 9.27: Average Response Time-640 Cameras

		Tree 1			Tree 2			Tree 3		
	$\#CFNs$	Original QG	Reconfigured QG	Adjusted QG	Original QG	Reconfigured QG	Adjusted QG	Original QG	Reconfigured QG	Adjusted QG
C_1	2	17893.13(s)	110761.98 (s)	110372.16(s)	13665.37 (s)	85654.04(s)	87186.551(s)	11461.92(s)	74312.64(s)	73986.962(s)
	10	2096.32 (s)	20942.178(s)	20931.013(s)	1569.181(s)	16133.482(s)	16431.57 (s)	1191.22(s)	13889.764(s)	13826.07(s)
	20	551.2033 (s)	12128.05(s)	12091.95(s)	424.13 (s)	9472.952 (s)	9518.887 (s)	176.101(s)	7954.08(s)	8293.632 (s)
	30	329.83(s)	8891.697(s)	9027.77(s)	262.483 (s)	7149.126 (s)	7190.922 (s)	62.515(s)	5175.71 (s)	5204.895(s)
	40	302.92(s)	5934.772(s)	5918.081(s)	240.319(s)	4787.316(s)	4998.159 (s)	56.418(s)	3900.824(s)	3909.7021 (s)
	50	69.214(s)	3319.92(s)	3316.03(s)	55.942(s)	2581.219 (s)	2581.29(s)	54.83(s)	2031.725(s)	2028.688(s)
	60	64.884 (s)	3292.262(s)	3287.433(s)	55.522(s)	2546.01(s)	2545.324 (s)	51.972(s)	2002.22(s)	1998.22(s)
	70	62.75(s)	1605.6(s)	1601.5(s)	49.745 (s)	1220.409(s)	1229.53 (s)	46.231(s)	857.52(s)	861.109(s)
	80	61.5863(s)	1560.22(s)	1561.247(s)	50.315 (s)	1200.162 (s)	1205.43(s)	45.381(s)	855.09(s)	851.939(s)
C_2	2	13962.15274 (s)	7944.76035 (s)	8033.04365(s)	10685.686 (s)	4800.492 (s)	4935.208(s)	9018.705(s)	4172.3332(s)	4216.63(s)
	10	1652.0861 (s)	1549.68309 (s)	1641.8859(s)	1243.6459(s)	1036.474572 (s)	1096.6073(s)	961.469(s)	911.886(s)	911.151(s)
	20	450.489277 (s)	749.250(s)	783.7165(s)	350.43551(s)	534.484 (s)	529.2255(s)	167.7087(s)	450.24127(s)	434.73716(s)
	30	275.717(s)	550.31831(s)	546.41222(s)	226.6059(s)	404.1327(s)	420.49945(s)	79.43598(s)	294.08049(s)	291.17107 (s)
	40	254.757(s)	368.861(s)	384.15935(s)	206.47166 (s)	272.49527(s)	272.28232(s)	72.15018(s)	267.41092(s)	258.195844(s)
	50	73.10519(s)	208.05995(s)	209.38504(s)	65.81181(s)	149.173 (s)	139.91797(s)	71.7377(s)	141.88163(s)	139.78202(s)
	60	68.59422(s)	167.49871(s)	168.21602(s)	65.27984(s)	134.16721(s)	135.91111(s)	67.54707(s)	87.18215(s)	78.87738(s)
	70	66.73695(s)	76.97734(s)	70.922317(s)	65.28715(s)	65.23832(s)	61.26103(s)	60.28496(s)	53.274312(s)	49.09479(s)
	80	65.91094(s)	62.88131(s)	60.193454 (s)	64.2745(s)	62.74545(s)	60.264678(s)	59.14714(s)	38.46183(s)	36.03381(s)
C_3	2	12213.317 (s)	5904.67(s)	5732.478(s)	9355.775(s)	4162.07296(s)	3948.692(s)	7917.496(s)	3619.531(s)	3528.017(s)
	10	1451.347 (s)	1343.193 (s)	1215.859(s)	1095.166(s)	899.258 (s)	845.477(s)	853.181(s)	791.901(s)	696.529(s)
	20	401.803 (s)	650.045(s)	479.311(s)	313.939(s)	464.396 (s)	455.3475(s)	158.952(s)	391.603(s)	374.616(s)
	30	248.154(s)	477.734(s)	469.541(s)	206.788(s)	351.655(s)	342.555(s)	81.129(s)	256.36(s)	251.33 (s)
	40	229.828 (s)	320.64952(s)	233.3175(s)	187.7999(s)	237.42 (s)	215.317 (s)	73.794(s)	233.267(s)	222.9269 (s)
	50	71.201(s)	181.339(s)	160.882(s)	63.1493(s)	130.58956(s)	121.08448(s)	73.7476(s)	124.48(s)	121.148(s)
	60	65.1322(s)	146.211(s)	125.5374(s)	62.60569 (s)	128.841(s)	117.56(s)	69.337(s)	76.944 (s)	68.8504 (s)
	70	63.501641(s)	67.85449(s)	61.429608(s)	60.07384(s)	62.749 (s)	58.26622(s)	61.932(s)	47.641285(s)	43.2913(s)
	80	62.74352(s)	60.07289(s)	56.98129(s)	60.5896(s)	59.53231(s)	55.50138(s)	60.94688(s)	37.61934(s)	35.01622(s)

Table 9.28: Average Distortion Due to Buffering-64 Cameras

		Tree 1			Tree 2			Tree 3		
	$\#CFNs$	Original QG	Reconfigured QG	Adjusted QG	Original QG	Reconfigured QG	Adjusted QG	Original QG	Reconfigured QG	Adjusted QG
C_1	2	30.62 (%)	47.76 (%)	47.76 (%)	29.79 (%)	47.69 (%)	25.2 (%)	47.38 (%)	47.37 (%)	47.39 (%)
	10	0 (%)	36.04 (%)	36.05 (%)	0 (%)	35.19 (%)	35.2 (%)	0 (%)	0 (%)	0 (%)
	20	0 (%)	12.5 (%)	12.5 (%)	0 (%)	10.35 (%)	11.29 (%)	0 (%)	0 (%)	0 (%)
	30	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	40	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	50	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	60	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	70	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	80	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
C_2	2	30.21 (%)	24.93 (%)	25.03 (%)	29.32 (%)	24.61 (%)	24.68 (%)	24.68 (%)	21.28 (%)	21.01 (%)
	10	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	20	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	30	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	40	0 (%)	0 (%)	0 (s)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	50	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	60	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	70	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	80	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
C_3	2	28.02 (%)	22.04 (%)	21.51 (%)	27.09 (%)	21.24 (%)	21.24 (%)	21.54 (%)	17.39 (%)	16.91 (%)
	10	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	20	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	30	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	40	0 (%)	0 (%)	0 (s)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	50	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	60	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	70	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	80	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)

and 9.32 show that by increasing the number of CFNs the average distortion due to buffering decreases. The reason is that increasing the number of fog nodes means more available processing resources close to the cameras. Video chunks are placed in a fog node's buffer upon its arrival. The fog node reads from the buffer and processes the video chunks. When the number of fog nodes increases there are fewer cameras associated with each fog node. Accordingly, the buffer size when there are more level 0 fog nodes and as a result the amount of time for video chunks to wait in a buffer until it is processed is smaller and this results in a reduction of

Table 9.29: Average Distortion Due to Buffering-128 Cameras

		Tree 1			Tree 2			Tree 3		
	$\#CFNs$	Original QG	Reconfigured QG	Adjusted QG	Original QG	Reconfigured QG	Adjusted QG	Original QG	Reconfigured QG	Adjusted QG
C_1	2	34.25 (%)	48.34 (%)	48.39 (%)	33.57 (%)	48.29 (%)	48.32 (%)	29.86 (%)	48.06 (%)	48.09 (%)
	10	0 (%)	39.66 (%)	39.82 (%)	0 (%)	39.27 (%)	39.54 (%)	0 (%)	37.17 (%)	37.43 (%)
	20	0 (%)	22.21 (%)	22.64 (%)	0 (%)	20.68 (%)	21.06 (%)	0 (%)	10.52 (%)	10.86 (%)
	30	0 (%)	0 (%)	0 (%)	(%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	40	0 (%)	0 (%)	0 (%)	(%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	50	0 (%)	0 (%)	0 (%)	(%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	60	0 (%)	0 (%)	0 (%)	(%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	70	0 (%)	0 (%)	0 (%)	(%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	80	0 (%)	0 (%)	0 (%)	(%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
C_2	2	33.98 (%)	32.23 (%)	32.45 (%)	33.28 (%)	31.26 (%)	31.57 (%)	29.52 (%)	28.87 (%)	29.88 (%)
	10	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	20	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	30	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	40	0 (%)	0 (%)	0 (s)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	50	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	60	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	70	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	80	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
C_3	2	32.14 (%)	30 (%)	27.94 (%)	33.66 (%)	28.92 (%)	27.47 (%)	27.15 (%)	26.22 (%)	24.28 (%)
	10	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	20	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	30	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	40	0 (%)	0 (%)	0 (s)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	50	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	60	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	70	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	80	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)

Table 9.30: Average Distortion Due to Buffering-256 Cameras

		Tree 1			Tree 2			Tree 3		
	$\#CFNs$	Original QG	Reconfigured QG	Adjusted QG	Original QG	Reconfigured QG	Adjusted QG	Original QG	Reconfigured QG	Adjusted QG
C_1	2	46.73 (%)	49.47 (%)	49.47 (%)	46.64 (%)	49.45 (%)	49.45 (%)	46.13 (%)	49.38 (%)	49.38 (%)
	10	22.05 (%)	47.19 (%)	23.6 (%)	20.58 (%)	47.11 (%)	47.11 (%)	12.34 (%)	46.72 (%)	46.72 (%)
	20	0 (%)	45.15 (%)	45.16 (%)	0 (%)	45.01 (%)	45.01 (%)	0 (%)	44.27 (%)	44.27 (%)
	30	0 (%)	43.39 (%)	43.4 (%)	0 (%)	43.39 (%)	43.39 (%)	0 (%)	41.19 (%)	41.28 (%)
	40	0 (%)	40.1 (%)	40.11 (%)	0 (%)	40.09 (%)	40.09 (%)	0 (%)	38.31 (%)	38.4 (%)
	50	0 (%)	32.3 (%)	32.35 (%)	0 (%)	31.61 (%)	31.61 (%)	0 (%)	27.55 (%)	27.63 (%)
	60	0 (%)	32.16 (%)	32.19 (%)	0 (%)	31.34 (%)	31.18 (%)	0 (%)	27.21 (%)	27.3 (%)
	70	0 (%)	13.34 (%)	13.41 (%)	0 (%)	11.2 (%)	11.37 (%)	0 (%)	0 (%)	0 (%)
	80	0 (%)	12.26 (%)	12.5 (%)	0 (%)	10.17 (%)	10.26 (%)	0 (%)	0 (%)	0 (%)
C_2	2	46.42 (%)	43.77 (%)	43.79 (%)	46.31 (%)	42.76 (%)	42.79 (%)	45.75 (%)	42.45(%)	42.49 (%)
	10	19.28 (%)	17.03 (%)	18.96 (%)	17.68 (%)	10.86 (%)	11.62 (%)	8.62 (%)	11.45(%)	11.37 (%)
	20	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	30	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	40	0 (%)	0 (%)	0 (s)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	50	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	60	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	70	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	80	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
C_3	2	45.97 (%)	42.93 (%)	42.86 (%)	45.85 (%)	41.78 (%)	41.75 (%)	45.22 (%)	41.44 (%)	41.4 (%)
	10	15.45 (%)	12.57 (%)	7.04 (%)	27.27 (%)	5.55 (%)	3 (%)	3.48 (%)	6.2 (%)	3.72 (%)
	20	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	30	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	40	0 (%)	0 (%)	0 (s)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	50	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	60	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	70	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	80	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)

the distortion due to buffering.

Effectiveness of Fog-Tree Hierarchy: Tree 3, representing the shortest tree, has the lowest response time as the result of providing more CFNs closer to data sources than the other two trees. With Tree 2 each fog node can accept up to three children while each fog node in Tree 3 can accept up to four children. By comparing the results in Tables 9.28, 9.29, 9.30, 9.30, 9.31, and 9.32, we can conclude that the fog-tree hierarchy has a good tolerance for processing the huge volume of data stream.

Table 9.31: Average Distortion Due to Buffering-512 Cameras

		Tree 1			Tree 2			Tree 3		
	$\#CFNs$	Original QG	Reconfigured QG	Adjusted QG	Original QG	Reconfigured QG	Adjusted QG	Original QG	Reconfigured QG	Adjusted QG
C_1	2	48.83 (%)	49.81 (%)	49.81 (%)	48.46 (%)	49.75(%)	49.76 (%)	48.18 (%)	49.71 (%)	49.71 (%)
	10	39.97 (%)	49 (%)	49 (%)	36.57 (%)	48.7(%)	48.71 (%)	64.32/2 (%)	48.42 (%)	48.13 (%)
	20	11.2 (%)	48.27 (%)	48.27 (%)	0 (%)	47.79 (%)	47.8 (%)	0 (%)	47.37 (%)	47.48(%)
	30	0 (%)	47.65 (%)	47.68(%)	0 (%)	47.07(%)	47.09 (%)	0 (%)	45.96 (%)	45.98 (%)
	40	0 (%)	46.48 (%)	46.47 (%)	0 (%)	45.63 (%)	45.82 (%)	0 (%)	44.64 (%)	44.65 (%)
	50	0 (%)	43.7 (%)	43.69 (%)	0 (%)	41.89 (%)	41.89 (%)	0 (%)	39.7 (%)	39.69 (%)
	60	0 (%)	43.65 (%)	43.64 (%)	0 (%)	41.78 (%)	41.78 (%)	0 (%)	39.55 (%)	39.55 (%)
	70	0 (%)	36.97(%)	36.94 (%)	0 (%)	32.84 (%)	32.98 (%)	0 (%)	25.54 (%)	25.67 (%)
	80	0 (%)	36.59 (%)	36.59 (%)	0 (%)	32.55 (%)	32.55 (%)	0 (%)	25.475 (%)	25.37(%)
C_2	2	48.53 (%)	47.43 (%)	47.46 (%)	48.7(%)	45.74(%)	45.86 (%)	47.7 (%)	45.09 (%)	45.14 (%)
	10	37.43 (%)	34.89 (%)	37.58 (%)	33.16 (%)	30.21 (%)	31.34(%)	27.63 (%)	27.47(%)	27.5 (%)
	20	1.35 (%)	22.64 (%)	23.95 (%)	0 (%)	11.45 (%)	11.2 (%)	0 (%)	3.96 (%)	2.5 (%)
	30	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	40	0 (%)	0 (%)	0 (s)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	50	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	60	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	70	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	80	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
C_3	2	48.33 (%)	46.57 (%)	46.47 (%)	47.82 (%)	45.12 (%)	44.87 (%)	47.39(%)	44.37(%)	44.24 (%)
	10	35.73 (%)	34.89 (%)	33.36 (%)	30.89 (%)	27.35 (%)	25.96 (%)	24.71 (%)	24.21 (%)	20.68 (%)
	20	0 (%)	6.28 (%)	7.64 (%)	0 (%)	5.88 (%)	5.22 (%)	0 (%)	0 (%)	0 (%)
	30	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	40	0 (%)	0 (%)	0 (s)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	50	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	60	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	70	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	80	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)

Table 9.32: Average Distortion Due to Buffering-640 Cameras

		Tree 1			Tree 2			Tree 3		
	$\#CFNs$	Original QG	Reconfigured QG	Adjusted QG	Original QG	Reconfigured QG	Adjusted QG	Original QG	Reconfigured QG	Adjusted QG
C_1	2	48.99 (%)	55.88 (%)	54.31 (%)	48.44 (%)	49.78 (%)	49.79 (%)	48.42 (%)	49.57 (%)	49.75 (%)
	10	41.35 (%)	49.14 (%)	49.13 (%)	38.42 (%)	48.88 (%)	48.9 (%)	34.62 (%)	48.7 (%)	48.69 (%)
	20	15.54 (%)	48.51 (%)	48.51 (%)	6.31 (%)	48.09 (%)	48.1 (%)	0 (%)	47.73 (%)	46.53 (%)
	30	0 (%)	46.97 (%)	48.01 (%)	0 (%)	47.48 (%)	47.19 (%)	0 (%)	46.51 (%)	46.53 (%)
	40	0 (%)	46.96 (%)	46.95 (%)	0 (%)	46.23 (%)	46.39 (%)	0 (%)	45.38 (%)	45.39 (%)
	50	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	60	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	70	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	80	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
C_2	2	48.7 (%)	47.66 (%)	47.75 (%)	48.3 (%)	46.23 (%)	46.34 (%)	47.96 (%)	45.65 (%)	45.7 (%)
	10	38.88 (%)	38.33 (%)	39.01 (%)	35.11 (%)	32.49 (%)	33.5 (%)	30.21 (%)	30.06 (%)	30.03 (%)
	20	6.93 (%)	25.77 (%)	26.95 (%)	0 (%)	15.9 (%)	15.64 (%)	0 (%)	9.36 (%)	8.04 (%)
	30	0 (%)	16.99 (%)	16.91 (%)	0 (%)	4.77 (%)	6.73 (%)	0 (%)	0 (%)	0 (%)
	40	0(%)	0.82 (%)	2.75 (%)	0 (%)	0(%)	0 (%)	0 (%)	0 (%)	0 (%)
	50	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	60	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	70	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	80	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
C_3	2	48.51 (%)	46.93 (%)	46.85 (%)	48.05 (%)	45.64 (%)	45.41 (%)	47.67 (%)	44.98 (%)	44.85 (%)
	10	37.27 (%)	36.51 (%)	35.14 (%)	32.93 (%)	29.77 (%)	28.54 (%)	27.32 (%)	26.98 (%)	23.83 (%)
	20	0.68 (%)	22 (%)	12.18 (%)	0(%)	13.89 (%)	10.02 (%)	0 (%)	3 (%)	1.08 (%)
	30	0 (%)	11.86 (%)	11.37 (%)	0 (%)	0 (%)	0(%)	0 (%)	0(%)	0 (%)
	40	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	50	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	60	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	70	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)
	80	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)	0 (%)

Effectiveness of Algorithms 8.1 and 8.2: From Tables 9.28, 9.29, 9.30, 9.30, 9.31, and 9.32 we see that the original query graph results in less distortion compared to the reconfigured and the adjusted query graph.

Comparison of Cost Functions: From Tables 9.28, 9.29, 9.30, 9.30, 9.31, and 9.32 we see that cost function C_3 has lower distortion due to buffering compared to the cost functions C_1 and C_2. The reason is that cost function C_3 maps query vertices of a given query graph to nodes in the upper level of the tree overlay network of fog nodes. Fog nodes in upper level of the

hierarchy of fog nodes have more processing power compared to fog nodes in the lower levels of the hierarchy.

9.3.5 Average Distortion Due to Network Latency

Tables 9.33, 9.34, 9.35, 9.36, and 9.37 show the average distortion due to buffering for processing of the original query graph, reconfigured query graph, and adjusted query graph as the number of fog nodes and vehicles vary. Moreover, tables 9.33, 9.34, 9.35, 9.36, and 9.37 compares the effect of Tree 1, Tree 2, and Tree 3 on the distortion due to network latency parameter. Tables 9.33, 9.34, 9.35, 9.36, and 9.37 shows that by increasing the number of level 0 fog nodes we can better cope with the network latency for a large number of data sources by increasing the number of fog nodes. The result of decreasing end-to-end delay decreases the average distortion.

Table 9.33: Average Distortion Due to Network Latency-64 Cameras

		Tree 1			Tree 2			Tree 3		
	#CFNs	Original QG	Reconfigured QG	Adjusted QG	Original QG	Reconfigured QG	Adjusted QG	Original QG	Reconfigured QG	Adjusted QG
C_1	2	48.32 (%)	46.83 (%)	46.65 (%)	48.39 (%)	46.93 (%)	46.81 (%)	49.09 (%)	47.58 (%)	47.57 (%)
	10	45.15 (%)	36.52 (%)	35.79 (%)	45.32 (%)	36.61 (%)	36.57 (%)	46.08 (%)	39.78 (%)	39.26 (%)
	20	42.83 (%)	29.98 (%)	29.08 (%)	43.64 (%)	31.51 (%)	30.82 (%)	43.79 (%)	32.18 (%)	32.11 (%)
	30	41.59 (%)	29.89 (%)	28.41 (%)	43.55 (%)	31.46 (%)	30.16 (%)	43.34 (%)	32.04 (%)	31.89 (%)
	40	39.41 (%)	28.68 (%)	28.36 (%)	41.9 (%)	31.47 (%)	30.79 (%)	42.72 (%)	32.02 (%)	31.95 (%)
	50	35.13 (%)	28.32 (%)	27.81 (%)	37.11 (%)	31.55 (%)	31.25 (%)	41.71 (%)	31.91 (%)	31.89 (%)
	60	36.34 (%)	30.52 (%)	30.38 (%)	38.24 (%)	31.65 (%)	31.48 (%)	42.32 (%)	33.17 (%)	33.14 (%)
	70	32.25 (%)	30.59 (%)	30.38 (%)	37.78 (%)	33.98 (%)	33.89 (%)	43.52 (%)	34.13 (%)	33.915 (%)
	80	27.92 (%)	30.88 (%)	30.64 (%)	37.4 (%)	34.36 (%)	34.26 (%)	40.33 (%)	34.7 (%)	34.53 (%)
C_2	2	48.76 (%)	47.29 (%)	46.57 (%)	48.8 (%)	47.37 (%)	46.71 (%)	49.33 (%)	4792 (%)	47.51 (%)
	10	46.4 (%)	38.46 (%)	35.44 (%)	46.53 (%)	38.54 (%)	36.25 (%)	47.09 (%)	41.26 (%)	38.99 (%)
	20	44.68 (%)	32.87 (%)	28.58 (%)	45.28 (%)	34.18 (%)	30.37 (%)	45.39 (%)	34.75 (%)	31.69 (%)
	30	43.76 (%)	32.79 (%)	27.89 (%)	45.22 (%)	34.13 (%)	29.69 (%)	45.06 (%)	34.63(%)	31.46 (%)
	40	42.15 (%)	31.75 (%)	27.82(%)	43.99 (%)	34.15 (%)	30.34 (%)	44.6 (%)	34.61 (%)	31.52 (%)
	50	38.97 (%)	31.45 (%)	27.28 (%)	40.44 (%)	34.21 (%)	30.8 (%)	43.85 (%)	34.5 (%)	31.46 (%)
	60	39.87 (%)	16.66 (%)	29.91 (%)	41.28(%)	34.29 (%)	31.04 (%)	44.3 (%)	3.58 (%)	32.74 (%)
	70	36.84 (%)	33.39 (%)	29.91 (%)	40.94 (%)	36.29 (%)	33.51 (%)	44.09 (%)	36.42 (%)	33.53 (%)
	80	33.62(%)	33.64 (%)	30.16 (%)	40.66 (%)	36.61 (%)	33.89 (%)	42.82 (%)	36.91 (%)	34.16 (%)
C_3	2	48.88 (%)	47.56 (%)	46.96 (%)	48.93 (%)	47.64 (%)	47.1 (%)	49.39 (%)	48.13 (%)	47.79 (%)
	10	46.76 (%)	39.63 (%)	37.08 (%)	46.88 (%)	39.7 (%)	37.79 (%)	47.38 (%)	42.14 (%)	40.23 (%)
	20	45.22 (%)	34.6 (%)	30.98 (%)	45.76 (%)	35.78 (%)	32.57 (%)	45.86 (%)	36.18 (%)	33.74 (%)
	30	44.39 (%)	34.53 (%)	30.37 (%)	45.7 (%)	35.73 (%)	31.96 (%)	45.56 (%)	36.17 (%)	33.54 (%)
	40	42.94 (%)	33.6 (%)	30.32 (%)	44.6 (%)	35.75 (%)	32.54 (%)	45.15 (%)	36.08 (%)	33.57 (%)
	50	40.09(%)	33.32 (%)	29.83 (%)	42.16 (%)	35.88 (%)	33.16 (%)	44.88 (%)	37.05 (%)	33.54 (%)
	60	40.89 (%)	35.01 (%)	32.16 (%)	41.85 (%)	37.67 (%)	35.35 (%)	44.69 (%)	37.79 (%)	34.68 (%)
	70	38.17 (%)	35.07 (%)	32.16 (%)	41.73 (%)	37.73 (%)	35.75 (%)	44.43 (%)	37.89 (%)	35.38 (%)
	80	35.27 (%)	35.29 (%)	32.39 (%)	41.6 (%)	37.96 (%)	35.79 (%)	43.55 (%)	38.23 (%)	35.93 (%)

Table 9.34: Average Distortion Due to Network Latency-128 Cameras

		Tree 1			Tree 2			Tree 3		
	$\#CFNs$	Original QG	Reconfigured QG	Adjusted QG	Original QG	Reconfigured QG	Adjusted QG	Original QG	Reconfigured QG	Adjusted QG
C_1	2	48.72 (%)	47.65 (%)	47.44 (%)	48.76 (%)	47.71 (%)	47.55 (%)	49.27 (%)	48.34 (%)	48.05 (%)
	10	46.82 (%)	43.45 (%)	43.1 (%)	46.9 (%)	43.47 (%)	43.33 (%)	47.31 (%)	44.43 (%)	44.1 (%)
	20	45.74 (%)	42.02 (%)	41.7 (%)	46.09 (%)	42.31 (%)	42.01 (%)	46.16 (%)	42.44 (%)	42.27 (%)
	30	45.25 (%)	42.01 (%)	41.59 (%)	46.05 (%)	42.3 (%)	41.89 (%)	45.96 (%)	42.41 (%)	42.22 (%)
	40	44.48 (%)	41.8 (%)	41.58 (%)	45.36 (%)	42.3 (%)	42.01 (%)	45.7 (%)	42.41 (%)	42.23 (%)
	50	43.29 (%)	41.74 (%)	41.5 (%)	43.8 (%)	42.32 (%)	42.09 (%)	45.29 (%)	42.49 (%)	42.39 (%)
	60	43.59 (%)	42.12 (%)	41.93 (%)	44.12 (%)	42.34 (%)	42.14 (%)	45.53 (%)	42.66 (%)	42.51 (%)
	70	42.64 (%)	42.13 (%)	41.93 (%)	43.99 (%)	42.45 (%)	42.05 (%)	45.42 (%)	42.66 (%)	42.51 (%)
	80	41.86 (%)	42.19 (%)	41.98 (%)	43.88 (%)	42.91 (%)	42.73 (%)	44.79 (%)	43 (%)	42.83 (%)
C_2	2	49.16 (%)	48.25 (%)	47.81 (%)	49.19 (%)	48.3 (%)	47.9 (%)	49.52 (%)	48.77 (%)	48.33 (%)
	10	47.91 (%)	45.14 (%)	44.09 (%)	47.97 (%)	45.16 (%)	44.3 (%)	48.24 (%)	45.87 (%)	44.95 (%)
	20	47.21 (%)	44.08 (%)	42.9 (%)	47.44 (%)	44.29 (%)	43.17 (%)	47.49 (%)	44.39 (%)	43.38 (%)
	30	46.89 (%)	44.07 (%)	42.8 (%)	47.42 (%)	44.29 (%)	43.06 (%)	47.36 (%)	44.37 (%)	43.34 (%)
	40	46.39 (%)	43.92 (%)	42.8 (%)	46.96 (%)	44.29 (%)	43.16 (%)	47.18 (%)	44.37 (%)	43.35 (%)
	50	45.6 (%)	43.87 (%)	85.45 (%)	45.94 (%)	44.3 (%)	43.23 (%)	46.92 (%)	44.39 (%)	43.48 (%)
	60	45.8 (%)	44.16 (%)	43.1 (%)	46.15 (%)	44.31 (%)	43.27 (%)	47.07 (%)	.4455 (%)	36.64 (%)
	70	45.19 (%)	44.17 (%)	43.1 (%)	46.06 (%)	44.4 (%)	43.19 (%)	47 (%)	44.55 (%)	43.59 (%)
	80	44.67 (%)	44.21 (%)	43.12 (%)	45.99 (%)	44.74 (%)	43.78 (%)	46.59 (%)	44.8 (%)	43.87 (%)
C_3	2	49.24 (%)	48.43 (%)	48.03 (%)	49.27 (%)	48.47 (%)	48.11 (%)	49.57 (%)	48.89 (%)	48.5 (%)
	10	48.12 (%)	45.63 (%)	44.69 (%)	48.18 (%)	45.65 (%)	44.87 (%)	48.42 (%)	46.28 (%)	45.46 (%)
	20	47.49 (%)	44.68 (%)	43.62 (%)	47.7 (%)	44.87 (%)	43.86 (%)	47.74 (%)	44.96 (%)	44.05 (%)
	30	47.2 (%)	44.67 (%)	43.53 (%)	47.68 (%)	44.86 (%)	43.76 (%)	47.62 (%)	44.94 (%)	44.01 (%)
	40	46.75 (%)	44.53 (%)	43.52 (%)	47.27 (%)	44.87 (%)	43.85 (%)	47.47 (%)	44.94 (%)	44.02 (%)
	50	46.05 (%)	44.49 (%)	43.45 (%)	46.35 (%)	44.88 (%)	43.92 (%)	47.23 (%)	44.96 (%)	44.14 (%)
	60	46.23 (%)	44.74 (%)	43.79 (%)	46.54 (%)	44.89 (%)	43.95 (%)	47.37 (%)	45.1 (%)	37.99 (%)
	70	45.67 (%)	44.75 (%)	43.79 (%)	46.46 (%)	44.97 (%)	43.88 (%)	47.3 (%)	45.1 (%)	44.24 (%)
	80	45.21 (%)	44.79 (%)	43.83 (%)	46.4 (%)	45.27 (%)	44.41 (%)	46.93 (%)	4.53 (%)	44.48 (%)

Table 9.35: Average Distortion Due to Network Latency-256 Cameras

		Tree 1			Tree 2			Tree 3		
	$\#CFNs$	Original QG	Reconfigured QG	Adjusted QG	Original QG	Reconfigured QG	Adjusted QG	Original QG	Reconfigured QG	Adjusted QG
C_1	2	49.67 (%)	49.48 (%)	49.46 (%)	49.68 (%)	49.49 (%)	49.48 (%)	49.8 (%)	49.57 (%)	49.57 (%)
	10	48.8 (%)	48.35 (%)	48.31 (%)	48.835 (%)	48.38 (%)	48.34 (%)	49.2 (%)	48.53 (%)	48.53 (%)
	20	48.47 (%)	48.05 (%)	48.01 (%)	48.51 (%)	48.05 (%)	47.97 (%)	48.91 (%)	48.19 (%)	48.19 (%)
	30	48.41 (%)	47.89 (%)	47.81 (%)	48.48 (%)	47.91 (%)	47.89 (%)	48.87 (%)	48.13 (%)	48.08 (%)
	40	48.4 (%)	47.87 (%)	47.74 (%)	47.9 (%)	47.84 (%)	47.79 (%)	48.85 (%)	48.12 (%)	47.93 (%)
	50	47.94 (%)	47.65 (%)	47.65 (%)	48 (%)	47.71 (%)	47.71 (%)	48.22 (%)	47.98 (%)	47.58 (%)
	60	47.89 (%)	47.65 (%)	47.56 (%)	47.95 (%)	47.7 (%)	47.52 (%)	48.27 (%)	47.73 (%)	47.58 (%)
	70	47.88 (%)	47.65 (%)	47.81 (%)	48.18 (%)	47.72 (%)	47.44 (%)	48.19 (%)	47.8 (%)	47.58 (%)
	80	47.85 (%)	47.65 (%)	47.38 (%)	47.92 (%)	47.72 (%)	48.38 (%)	48.38 (%)	47.78 (%)	47.63 (%)
C_2	2	49.84 (%)	49.69 (%)	49.68 (%)	49.85 (%)	49.7 (%)	49.69 (%)	49.88 (%)	49.74 (%)	49.72 (%)
	10	49.58 (%)	49.03 (%)	49.02 (%)	49.58 (%)	49.02 (%)	49.03 (%)	49.69 (%)	49.15 (%)	49.14 (%)
	20	49.5 (%)	49.02 (%)	49.02 (%)	49.5 (%)	49.02 (%)	49.02 (%)	49.69 (%)	49.05 (%)	49.05(%)
	30	49.5 (%)	49.02 (%)	49.02(%)	49.5 (%)	49.01 (%)	49.01 (%)	49.67 (%)	49.05 (%)	49.04 (%)
	40	49.5 (%)	49.02 (%)	49.02(%)	49.51 (%)	49.01 (%)	48.89 (%)	49.67 (%)	49.04 (%)	48.88 (%)
	50	49.48 (%)	49.02 (%)	49.02 (%)	49.51 (%)	49.01 (%)	48.87 (%)	49.58 (%)	49.04 (%)	48.87 (%)
	60	49.48 (%)	49.01 (%)	49.01 (%)	49.5 (%)	49.01 (%)	48.87 (%)	49.58 (%)	49.01 (%)	48.87 (%)
	70	49.48 (%)	49.02 (%)	49.02 (%)	49.52 (%)	49.01 (%)	48.87 (%)	49.58 (%)	49.01(%)	48.87 (%)
	80	49.48 (%)	49.01 (%)	48.98 (%)	49.52 (%)	49.01 (%)	48.87 (%)	49.58 (%)	49.01 (%)	48.87 (%)
C_3	2	49.85 (%)	49.69 (%)	49.68 (%)	49.86 (%)	49.74 (%)	49.69 (%)	49.91 (%)	49.76 (%)	49.74 (%)
	10	49.58 (%)	49.03 (%)	49.02 (%)	49.58 (%)	49.04 (%)	49.02 (%)	49.73 (%)	49.13 (%)	49.11 (%)
	20	49.58 (%)	49.02 (%)	49.02(%)	49.58 (%)	49.03 (%)	49.02 (%)	49.73 (%)	49.12 (%)	49.09 (%)
	30	49.58 (%)	49.02 (%)	49.02 (%)	49.57 (%)	49.02(%)	49.02 (%)	49.69 (%)	49.11 (%)	49.02 (%)
	40	49.58 (%)	49.02 (%)	49.01 (%)	49.58 (%)	49.02 (%)	49.02 (%)	49.69 (%)	49.05 (%)	49.01 (%)
	50	49.49 (%)	49.02 (%)	49.01 (%)	49.58 (%)	49.02 (%)	49.01 (%)	49.69 (%)	49.03 (%)	49.01 (%)
	60	49.49 (%)	49.01 (%)	49.01 (%)	49.58 (%)	49.02 (%)	49.98(%)	49.68 (%)	49.02 (%)	49.01 (%)
	70	49.49 (%)	49.01 (%)	49.02 (%)	49.58 (%)	49.02(%)	48.97 (%)	49.67 (%)	49.02 (%)	49.01 (%)
	80	49.49 (%)	49.01 (%)	48.98(%)	49.58 (%)	49.02(%)	49.01 (%)	49.66 (%)	49.02 (%)	49.01 (%)

Table 9.36: Average Distortion Due to Network Latency-512 Cameras

		Tree 1			Tree 2			Tree 3		
	$\#CFNs$	Original QG	Reconfigured QG	Adjusted QG	Original QG	Reconfigured QG	Adjusted QG	Original QG	Reconfigured QG	Adjusted QG
C_1	2	98.77 (%)	96.2 (%)	91.76 (%)	98.96 (%)	96.77 (%)	94.2 (%)	99.38 (%)	97.1 (%)	96.37(%)
	10	97.07 (%)	87.77 (%)	73.82 (%)	97.23 (%)	89.65 (%)	84.49 (%)	98.13 (%)	90.72 (%)	88.57 (%)
	20	96.99 (%)	84.39 (%)	69.27 (%)	87.73 (%)	87.73 (%)	81.74 (%)	97.88 (%)	88.59 (%)	85.95 (%)
	30	96.56 (%)	84.23 (%)	66.12 (%)	97.09 (%)	86.7 (%)	80.23 (%)	97.84 (%)	88.21 (%)	85.11(%)
	40	96.56 (%)	84.23 (%)	65.08 (%)	96.62 (%)	86.22 (%)	79.33 (%)	97.65 (%)	88.13 (%)	83.96(%)
	50	96.56 (%)	82.79 (%)	63.65 (%)	96.69 (%)	85.46 (%)	78.52 (%)	97.72 (%)	87.24 (%)	81.77 (%)
	60	96.05 (%)	82.6 (%)	61.62 (%)	96.65 (%)	85.35 (%)	76.71 (%)	97.55 (%)	85.63 (%)	81.24 (%)
	70	96.04 (%)	82.6 (%)	61.61 (%)	96.83 (%)	85.45 (%)	75.98 (%)	97.27 (%)	86.08 (%)	81.22(%)
	80	96.02 (%)	82.6 (%)	59.51 (%)	96.99 (%)	85.45 (%)	77.47 (%)	97.16 (%)	87.68 (%)	81.61 (%)
C_2	2	49.97 (%)	49.9 (%)	49.79 (%)	49.97 (%)	49.91 (%)	49.87 (%)	49.98 (%)	49.92 (%)	49.91 (%)
	10	49.93 (%)	49.69 (%)	49.34 (%)	49.95 (%)	49.74 (%)	49.61 (%)	49.96 (%)	49.73 (%)	49.71 (%)
	20	49.93 (%)	49.58 (%)	48.99 (%)	49.93 (%)	49.66 (%)	49.46 (%)	49.95 (%)	49.71 (%)	49.64 (%)
	30	49.93 (%)	49.58(%)	48.99 (%)	49.94 (%)	49.66 (%)	49.46 (%)	49.95 (%)	49.68 (%)	49.54 (%)
	40	49.93 (%)	49.58(%)	48.99 (%)	49.93 (%)	49.65 (%)	49.43 (%)	49.95 (%)	49.68 (%)	49.53 (%)
	50	49.92 (%)	49.58 (%)	48.99 (%)	49.93 (%)	49.63 (%)	49.43 (%)	49.95 (%)	49.67 (%)	49.53 (%)
	60	49.92 (%)	49.57 (%)	48.99 (%)	49.93 (%)	49.63 (%)	49.43(%)	49.95 (%)	49.65 (%)	49.53 (%)
	70	49.92 (%)	49.56 (%)	48.98 (%)	49.93 (%)	49.63 (%)	49.43(%)	49.94 (%)	49.65 (%)	49.53 (%)
	80	49.92 (%)	49.56 (%)	48.98 (%)	49.93 (%)	49.63 (%)	49.43 (%)	49.94 (%)	49.61 (%)	49.53 (%)
C_3	2	49.97 (%)	49.91 (%)	49.81 (%)	49.98 (%)	49.92 (%)	49.89 (%)	49.99 (%)	49.93 (%)	49.9 (%)
	10	49.94 (%)	49.72 (%)	49.31 (%)	49.93 (%)	49.76 (%)	49.64 (%)	49.96 (%)	49.76 (%)	49.76 (%)
	20	49.94 (%)	49.69 (%)	49.3 (%)	49.92 (%)	49.73 (%)	49.58 (%)	49.96 (%)	49.76 (%)	49.75 (%)
	30	49.93 (%)	49.64 (%)	49.09 (%)	49.94 (%)	49.66 (%)	49.58 (%)	49.95 (%)	49.76 (%)	49.75 (%)
	40	49.93 (%)	49.64 (%)	49.08 (%)	49.92 (%)	49.66 (%)	49.57 (%)	49.92 (%)	49.75 (%)	49.63 (%)
	50	49.93 (%)	49.63 (%)	49.07 (%)	49.92 (%)	49.67 (%)	49.48 (%)	49.91 (%)	49.74 (%)	49.63 (%)
	60	49.92 (%)	49.6 (%)	49.07 (%)	49.92 (%)	49.66 (%)	49.48(%)	49.91 (%)	49.67 (%)	49.63 (%)
	70	49.92 (%)	49.6(%)	49.07 (%)	49.92 (%)	49.55 (%)	49.48 (%)	49.91 (%)	49.67 (%)	49.62 (%)
	80	49.92 (%)	49.6 (%)	49.07 (%)	49.94 (%)	49.53 (%)	49.48 (%)	49.9 (%)	49.66 (%)	49.63 (%)

Table 9.37: Average Distortion Due to Network Latency-640 Cameras

		Tree 1			Tree 2			Tree 3		
	$\#CFNs$	Original QG	Reconfigured QG	Adjusted QG	Original QG	Reconfigured QG	Adjusted QG	Original QG	Reconfigured QG	Adjusted QG
C_1	2	49.95 (%)	49.85 (%)	49.68 (%)	49.96 (%)	49.87(%)	49.81 (%)	49.97 (%)	49.89 (%)	49.87 (%)
	10	49.88 (%)	49.52 (%)	48.99 (%)	49.89 (%)	49.6 (%)	49.4 (%)	49.91 (%)	49.64 (%)	49.56(%)
	20	49.88 (%)	49.44 (%)	48.81 (%)	49.87 (%)	49.64 (%)	49.44 (%)	49.91 (%)	49.56(%)	49.45 (%)
	30	49.86 (%)	49.39 (%)	48.69 (%)	49.88 (%)	49.48 (%)	49.23 (%)	49.91 (%)	49.56 (%)	49.45 (%)
	40	49.86 (%)	49.39 (%)	48.69 (%)	49.87 (%)	49.48 (%)	49.07 (%)	49.21 (%)	49.59 (%)	49.38 (%)
	50	49.86 (%)	49.33 (%)	48.6 (%)	49.87 (%)	49.48 (%)	49.08 (%)	49.93 (%)	49.46 (%)	49.29 (%)
	60	49.84 (%)	49.33 (%)	48.55 (%)	49.87 (%)	49.44 (%)	49.07 (%)	49.9 (%)	49.46 (%)	49.29 (%)
	70	49.84 (%)	49.33 (%)	48.52 (%)	49.87 (%)	49.44 (%)	49.05 (%)	49.89 (%)	49.46 (%)	49.29 (%)
	80	49.84 (%)	49.33 (%)	48.47 (%)	49.88 (%)	49.44 (%)	49.13 (%)	49.89 (%)	49.52 (%)	49.29 (%)
C_2	2	49.98 (%)	49.92 (%)	49.84 (%)	49.98 (%)	49.94 (%)	49.91 (%)	49.99 (%)	49.94 (%)	49.93 (%)
	10	49.95 (%)	49.77 (%)	49.51 (%)	49.95 (%)	49.80 (%)	49.71 (%)	49.97 (%)	49.82 (%)	49.71 (%)
	20	49.95 (%)	49.73 (%)	49.42 (%)	49.94 (%)	49.78 (%)	49.66 (%)	49.96 (%)	49.78 (%)	49.66 (%)
	30	49.94 (%)	49.7 (%)	49.36 (%)	49.95 (%)	49.77 (%)	49.58 (%)	49.96 (%)	49.78 (%)	49.66 (%)
	40	49.94 (%)	49.7 (%)	49.35 (%)	49.94 (%)	49.76 (%)	49.57 (%)	49.96 (%)	49.77 (%)	49.58 (%)
	50	49.94 (%)	49.68 (%)	49.32 (%)	49.94 (%)	49.76 (%)	49.57 (%)	49.97 (%)	49.75 (%)	49.58 (%)
	60	49.83 (%)	49.68 (%)	49.29 (%)	49.94 (%)	49.76 (%)	49.55 (%)	49.96 (%)	49.73(%)	49.55 (%)
	70	49.83 (%)	49.67 (%)	49.29 (%)	49.95 (%)	49.76 (%)	49.55 (%)	49.59 (%)	49.74 (%)	49.55 (%)
	80	49.83 (%)	49.66 (%)	49.24 (%)	49.95 (%)	49.71 (%)	49.58 (%)	49.95 (%)	49.71 (%)	49.56 (%)
C_3	2	49.98 (%)	49.93 (%)	49.86 (%)	49.98 (%)	49.94 (%)	49.92 (%)	49.99 (%)	49.95 (%)	49.94 (%)
	10	49.95 (%)	49.8 (%)	49.57 (%)	49.96 (%)	49.83 (%)	49.74 (%)	49.97(%)	49.84 (%)	49.82 (%)
	20	49.95 (%)	49.76 (%)	49.49 (%)	49.95 (%)	49.81 (%)	49.73 (%)	49.97 (%)	49.82 (%)	49.76 (%)
	30	49.95 (%)	49.74 (%)	49.44 (%)	49.96 (%)	49.78 (%)	49.68 (%)	49.97 (%)	49.81 (%)	49.75 (%)
	40	49.95 (%)	49.74 (%)	49.42 (%)	49.95 (%)	49.78 (%)	49.68 (%)	49.96 (%)	49.8 (%)	49.74 (%)
	50	49.95 (%)	49.74 (%)	49.4 (%)	49.95 (%)	49.76 (%)	49.68 (%)	49.96 (%)	49.79 (%)	49.86 (%)
	60	49.94 (%)	49.71 (%)	49.38 (%)	49.95 (%)	49.76 (%)	49.67 (%)	49.96 (%)	49.76 (%)	49.66 (%)
	70	49.94 (%)	49.71 (%)	49.38 (%)	49.96 (%)	49.74 (%)	49.63 (%)	49.96 (%)	49.77 (%)	49.7 (%)
	80	49.94 (%)	49.71 (%)	49.33 (%)	49.95 (%)	49.74 (%)	49.63 (%)	49.96 (%)	49.74 (%)	49.69 (%)

Chapter 10

Conclusion

In a nutshell, this thesis addresses some of the limitations of current architectures for video streaming and IoT applications based on nearby computing resources e.g., cloudlet, fog. The contributions of this thesis can be divided into two parts as described below:

1. In the first part of this thesis we presented the following contribution:

    - A framework (Coping Flash Crowd) for multi-channel P2P live video streaming that provides de-centralized mechanisms for handling flash crowds that includes incentive mechanism, load balancing mechanisms, and cross-channel help among the peers for live video streaming in multi-channel P2P systems. The performance evaluation results demonstrated that CFC decreases the redundant traffic between autonomous systems. It can cope the flash crowd phenomena in P2P Network, and reduce the startup delay because in live video stream networks a newcomer expects to watch a video immediately.
    - We extended Coping Flash Crowd framework to increase the quality of service for multi-channel P2P live video streaming systems. The extended Coping Flash Crowd framework considers mechanisms for coping with the flash crowd phenomena alongside the load balancing, traffic localization and network coding. Our performance results demonstrated that the redundant traffic between autonomous systems decreases significantly. This provides strong evidence of the effectiveness of using networking coding in extended Coping Flash Crowd framework.
    - A dynamic partitioning framework was proposed that considers network conditions and the amount of data being transferred, shows how a cloudlet mesh can be used, and presents performance results that illustrate the advantages of a dynamic partitioning strategy and a cloudlet mesh.

2. In the second part of this thesis we addressed the challenges of the current architecture for IoT applications. Currently the architecture for IoT applications assumes that data streams are sent to a cloud for analysis in order to determine a response. The major problem relying only on the cloud is the latency. The latency is due to the long network distance from the IoT devices to the cloud, which could lead to poor quality of service. Moreover, the network core could be overwhelmed [111] due to huge amount

of data that is generated by IoT devices. Accordingly, the use of a cloud is unsuitable for applications which require a high level of interaction with users and applications with real-time decision making requirements. This thesis we proposed a query operator placement algorithm which supports: Real-time aggregation and analysis of data streams from geographically distributed data sources; Combining more operators into one query vertex instead of having different query vertices for each operators helps to save more bandwidth and reduce the transmission costs; Optimized operator placement and operator reuse; Provide scalability for distributed data stream management systems.

10.1 Future Work

An adaptive approach for placement of query operators has been left for the future. In other words, a mechanism is required to periodically collects processing loads on fog nodes and if a processing loads on a fog node that hosts a query operator is higher that a predefined threshold then tries to find a new host for the query operators. Future work considers how the change of communication load and processing load in the networked fog nodes to find the hosts for query operators.

Bibliography

[1] Apache storm, apache soft ware foundation. 2014.

[2] Daniel J Abadi, Yanif Ahmad, Magdalena Balazinska, Ugur Cetintemel, Mitch Cherniack, Jeong-Hyon Hwang, Wolfgang Lindner, Anurag S Maskey, Alexander Rasin, Esther Ryvkina, Nesime Tatbul, Ying Xing, and Stan Zdonik. The Design of the Borealis Stream Processing Engine. In *Second Biennial Conference on Innovative Data Systems Research (CIDR 2005)*, Asilomar, CA, January 2005.

[3] A. Abdou, A. Matrawy, and P. C. van Oorschot. Accurate one-way delay estimation with reduced client trustworthiness. *IEEE Communications Letters*, 19(5):735–738, May 2015.

[4] Yanif Ahmad and Uğur Çetintemel. Network-aware query processing for stream-based applications. In *Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30*, VLDB '04, pages 456–467. VLDB Endowment, 2004.

[5] Nadeem Akhtar. Statistical data analysis of continuous streams using stream dsms. 2011.

[6] Nadeem Akhtar, Faraz Khan, and Mohammed A Qadeer. Information processing using data stream management system on jamdroid. In *Proceedings of the International Conference on Advances in Computing, Communication and Control*, ICAC3 '09, pages 208–213, New York, NY, USA, 2009. ACM.

[7] G. A. Akpakwu, B. J. Silva, G. P. Hancke, and A. M. Abu-Mahfouz. A survey on 5g networks for the internet of things: Communication technologies and challenges. *IEEE Access*, 6:3619–3647, 2018.

[8] Y. Al-Dhuraibi, F. Paraiso, N. Djarallah, and P. Merle. Elasticity in cloud computing: State of the art and research challenges. *IEEE Transactions on Services Computing*, PP(99):1–1, 2017.

[9] Muhammad Intizar Ali, Naomi Ono, Mahedi Kaysar, Keith Griffin, and Alessandra Mileo. A semantic processing framework for iot-enabled communication systems. In Marcelo Arenas, Oscar Corcho, Elena Simperl, Markus Strohmaier, Mathieu d'Aquin, Kavitha Srinivas, Paul Groth, Michel Dumontier, Jeff Heflin, Krishnaprasad Thirunarayan, and Steffen Staab, editors, *The Semantic Web - ISWC 2015*. Springer International Publishing, 2015.

[10] G. Andrienko, N. Andrienko, P. Jankowski, D. Keim, M. J. Kraak, A. MacEachren, and S. Wrobel. Geovisual analytics for spatial decision support: Setting the research agenda. *Int. J. Geogr. Inf. Sci.*, 21(8):839–857, January 2007.

[11] Arvind Arasu, Brian Babcock, Shivnath Babu, Mayur Datar, Keith Ito, Itaru Nishizawa, Justin Rosenstein, and Jennifer Widom. Stream: The stanford stream data manager (demonstration description). In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, SIGMOD '03, New York, NY, USA, 2003. ACM.

[12] Arvind Arasu, Shivnath Babu, and Jennifer Widom. The cql continuous query language: Semantic foundations and query execution. *The VLDB Journal*, 15(2), June 2006.

[13] M. H. Asghar, A. Negi, and N. Mohammadzadeh. Principle application and vision in internet of things (iot). In *International Conference on Computing, Communication Automation*, pages 427–431, May 2015.

[14] Shivnath Babu and Jennifer Widom. Continuous queries over data streams. *SIGMOD Rec.*, 30(3):109–120, September 2001.

[15] Nathan Backman, Karthik Pattabiraman, Rodrigo Fonseca, and Ugur Cetintemel. C-mr: Continuously executing mapreduce workflows on multi-core processors. In *Proceedings of Third International Workshop on MapReduce and Its Applications Date*, MapReduce '12, pages 1–8, New York, NY, USA, 2012. ACM.

[16] M. J. C. Baculo, C. S. Marzan, R. de Dios Bulos, and C. Ruiz. Geospatial-temporal analysis and classification of criminal data in manila. In *2017 2nd IEEE International Conference on Computational Intelligence and Applications (ICCIA)*, pages 6–11, Sept 2017.

[17] Rajesh Krishna Balan, Darren Gergle, Mahadev Satyanarayanan, and James Herbsleb. Simplifying cyber foraging for mobile devices. In *Proceedings of the 5th international conference on Mobile systems, applications and services*, MobiSys '07, 2007.

[18] Rajesh Krishna Balan, Mahadev Satyanarayanan, So Young Park, and Tadashi Okoshi. Tactics-based remote execution for mobile computing. In *Proceedings of the 1st International Conference on Mobile Systems, Applications and Services*, MobiSys '03, pages 273–286, New York, NY, USA, 2003. ACM.

[19] L. Baldesi, L. Maccari, and R. Lo Cigno. Improving p2p streaming in community-lab through local strategies. In *2014 IEEE 10th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, pages 33–39, Oct 2014.

[20] E. Balevi and R. D. Gitlin. Optimizing the number of fog nodes for cloud-fog-thing networks. *IEEE Access*, 6:11173–11183, 2018.

[21] Ingmar Baumgart, Bernhard Heep, and Stephan Krause. OverSim: A flexible overlay network simulation framework. In *IEEE GI, USA*, 2007.

[22] N. Bayat and H. Lutfiyya. Coping with flash crowd in multi-channel live video streaming for peer-to-peer networks. In *Mobile and Wireless Networking (MoWNeT), 2013 International Conference on Selected Topics in*, Aug 2013.

[23] N. Bayat and H. Lutfiyya. Mc-skynet: Mobile-cloud dynamic partitioning for mobile cloud applications. In *2014 IEEE Symposium on Computers and Communications (ISCC)*, volume Workshops, pages 1–7, June 2014.

[24] Navid Bayat, Behzad Akbari, Hamid R. Rabiee, and Mostafa Salehi. Locality aware p2p overlay architecture for live video streaming. In *Telecommunications (IST), 2012 Sixth International Symposium on*, 2012.

[25] I. Bermudez, M. Mellia, and M. Meo. Investigating overlay topologies and dynamics of p2p-tv systems: The case of sopcast. *IEEE Journal on Selected Areas in Communications*, 29(9):1863–1871, October 2011.

[26] Alain Biem, Eric Bouillet, Hanhua Feng, Anand Ranganathan, Anton Riabov, Olivier Verscheure, Haris Koutsopoulos, and Carlos Moran. Ibm infosphere streams for scalable, real-time, intelligent transportation services. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, SIGMOD '10, pages 1093–1104, New York, NY, USA, 2010. ACM.

[27] Hans L. Bodlaender and Tom C. van der Zanden. Improved lower bounds for graph embedding problems. *CoRR*, abs/1610.09130, 2016.

[28] Thomas Bonald, Laurent Massoulié, Fabien Mathieu, Diego Perino, and Andrew Twigg. Epidemic live streaming: Optimal performance trade-offs. In *Proceedings of the 2008 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '08, pages 325–336, New York, NY, USA, 2008. ACM.

[29] Philippe Bonnet, Johannes Gehrke, and Praveen Seshadri. Towards sensor database systems. In *Proceedings of the Second International Conference on Mobile Data Management*, MDM '01, pages 3–14, London, UK, UK, 2001. Springer-Verlag.

[30] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*. ACM, 2012.

[31] A. Botta, W. de Donato, V. Persico, and A. Pescap. On the integration of cloud computing and internet of things. In *2014 International Conference on Future Internet of Things and Cloud*, pages 23–30, Aug 2014.

[32] A. Brogi and S. Forti. Qos-aware deployment of iot applications through the fog. *IEEE Internet of Things Journal*, 4(5):1185–1192, Oct 2017.

[33] Rodrigo N. Calheiros, Rajiv Ranjan, Anton Beloglazov, César A. F. De Rose, and Rajkumar Buyya. Cloudsim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Softw. Pract. Exper.*, 41(1):23–50, January 2011.

[34] K. Calvert, J. Eagan, S. Merugu, A. Namjoshi, J. Stasko, and E. Zegura. Extending and enhancing gt-itm. In *ACM SIGCOMM workshop on Models, methods and tools for reproducible network research*, MoMeTools '03, pages 23–27, 2003.

[35] Valeria Cardellini, Vincenzo Grassi, Francesco Lo Presti, and Matteo Nardelli. Optimal operator placement for distributed stream processing applications. In *Proceedings of the 10th ACM International Conference on Distributed and Event-based Systems*, DEBS '16, pages 69–80, New York, NY, USA, 2016. ACM.

[36] Uğur Çetintemel, Daniel Abadi, Yanif Ahmad, Hari Balakrishnan, Magdalena Balazinska, Mitch Cherniack, Jeong-Hyon Hwang, Samuel Madden, Anurag Maskey, Alexander Rasin, Esther Ryvkina, Mike Stonebraker, Nesime Tatbul, Ying Xing, and Stan Zdonik. *The Aurora and Borealis Stream Processing Engines*. Springer Berlin Heidelberg, 2016.

[37] Sirish Chandrasekaran, Owen Cooper, Amol Deshpande, Michael J. Franklin, Joseph M. Hellerstein, Wei Hong, Sailesh Krishnamurthy, Samuel R. Madden, Fred Reiss, and Mehul A. Shah. Telegraphcq: Continuous dataflow processing. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, SIGMOD '03, pages 668–668, New York, NY, USA, 2003. ACM.

[38] Jianjun Chen, David J. DeWitt, Feng Tian, and Yuan Wang. Niagaracq: A scalable continuous query system for internet databases. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, SIGMOD '00, pages 379–390, New York, NY, USA, 2000. ACM.

[39] Liang Chen, K. Reddy, and G. Agrawal. Gates: a grid-based middleware for processing distributed data streams. In *Proceedings. 13th IEEE International Symposium on High performance Distributed Computing, 2004.*, pages 192–201, June 2004.

[40] Yishuai Chen, Baoxian Zhang, and Changjia Chen. Modeling and performance analysis of p2p live streaming systems under flash crowds. In *Communications (ICC), IEEE International Conference on*, june 2011.

[41] M. Chiang and T. Zhang. Fog and iot: An overview of research opportunities. *IEEE Internet of Things Journal*, 3(6):854–864, Dec 2016.

[42] Kenjiro Cho, Kensuke Fukuda, Hiroshi Esaki, and Akira Kato. Observing slow crustal movement in residential user traffic. In *Proceedings of the 2008 ACM CoNEXT Conference*, CoNEXT '08, pages 12:1–12:12, New York, NY, USA, 2008. ACM.

[43] Philip A. Chou and Yunnan Wu. Network coding for the internet and wireless networks. *Signal Processing Magazine*, September 2007.

[44] S. Choy, B. Wong, G. Simon, and C. Rosenberg. The brewing storm in cloud gaming: A measurement study on cloud to end-user latency. In *Network and Systems Support for Games (NetGames), 2012 11th Annual Workshop on*, pages 1–6, Nov 2012.

[45] Baochun Li Chuan Wu. On meeting p2p streaming bandwidth demand with limited supplies, 2008.

[46] Byung-Gon Chun, Sunghwan Ihm, Petros Maniatis, Mayur Naik, and Ashwin Patti. Clonecloud: elastic execution between mobile device and cloud. In *Proceedings of the sixth conference on Computer systems*, EuroSys '11, pages 301–314, 2011.

[47] Tein-Yaw Chung, Chih-Cheng Wang, Yung-Mu Chen, and Yang-Hui Chang. Pnecos: A peer-to-peer network coding streaming system. In *Sensor Networks, Ubiquitous and Trustworthy Computing, 2008. SUTC '08. IEEE International Conference on*, 2008.

[48] D. Ciullo, M. A. Garcia, A. Horvath, E. Leonardi, M. Mellia, D. Rossi, M. Telek, and P. Veglia. Network awareness of p2p live streaming applications: A measurement study. *IEEE Transactions on Multimedia*, 12(1):54–63, Jan 2010.

[49] D. Ciullo, M.A. Garcia, A. Horvath, E. Leonardi, M. Mellia, D. Rossi, M. Telek, and P. Veglia. Network awareness of p2p live streaming applications: A measurement study. *Multimedia, IEEE Transactions on*, 2010.

[50] R. G. Clegg, R. Landa, D. Griffin, E. Mykoniati, and M. Rio. Performance of locality-aware topologies for peer-to-peer live streaming. *IET Software*, 3(6):470–479, December 2009.

[51] Thomas Cooper. Proactive scaling of distributed stream processing work flows using workload modelling: Doctoral symposium. In *Proceedings of the 10th ACM International Conference on Distributed and Event-based Systems*, DEBS '16, pages 410–413, New York, NY, USA, 2016. ACM.

[52] Chuck Cranor, Theodore Johnson, Oliver Spataschek, and Vladislav Shkapenyuk. Gigascope: A stream database for network applications. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, SIGMOD '03, pages 647–651, New York, NY, USA, 2003. ACM.

[53] Eduardo Cuervo, Aruna Balasubramanian, Dae-ki Cho, Alec Wolman, Stefan Saroiu, Ranveer Chandra, and Paramvir Bahl. Maui: making smartphones last longer with code offload. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, MobiSys '10, pages 49–62, 2010.

[54] Michael Cusumano. Cloud computing and saas as new computing platforms. *Commun. ACM*, 53(4):27–29, April 2010.

[55] Abhinandan Das, Johannes Gehrke, and Mirek Riedewald. Approximate join processing over data streams. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, SIGMOD '03, pages 40–51, New York, NY, USA, 2003. ACM.

[56] S. K. Datta, C. Bonnet, and J. Haerri. Fog computing architecture to enable consumer centric internet of things services. In *2015 International Symposium on Consumer Electronics (ISCE)*, pages 1–2, June 2015.

[57] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. In *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, OSDI'04, pages 10–10, Berkeley, CA, USA, 2004. USENIX Association.

[58] K. Dolui and S. K. Datta. Comparison of edge computing implementations: Fog computing, cloudlet and mobile edge computing. In *2017 Global Internet of Things Summit (GIoTS)*, pages 1–6, June 2017.

[59] A. Draghici, M. Sandu-Popa, R. Deaconescu, and N. Tapus. A peer-to-peer swarm creation and management framework. In *2010 12th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, pages 545–548, Sep. 2010.

[60] R. Eidenbenz and T. Locher. Task allocation for distributed stream processing. In *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, pages 1–9, April 2016.

[61] Manal El Dick, Esther Pacitti, Reza Akbarinia, and Bettina Kemme. Review: Building a peer-to-peer content distribution network with high performance, scalability and robustness. *Inf. Syst.*

[62] A. Ephremides, S. Jaggi, T. Ho, B. Shrader, and S. Y. Chung. Network coding. *Journal of Communications and Networks*, 10(4):367–370, Dec 2008.

[63] Robert Epstein, Michael Stonebraker, and Eugene Wong. Distributed query processing in a relational data base system. In *Proceedings of the 1978 ACM SIGMOD International Conference on Management of Data*, SIGMOD '78, pages 169–180, New York, NY, USA, 1978. ACM.

[64] Dave Evans. The internet of things how the next evolution of the internet is changing everything. In *Technical report*. CISCO IBSG, 2011.

[65] Neng Fan and Panos M. Pardalos. Linear and quadratic programming approaches for the general graph partitioning problem. *J. of Global Optimization*, 48(1):57–71, September 2010.

[66] Debessay Fesehaye, Yunlong Gao, Klara Nahrstedt, and Guijun Wang. Impact of cloudlets on interactive mobile cloud applications. In *Proceedings of the 2012 IEEE 16th International Enterprise Distributed Object Computing Conference*, EDOC '12, pages 123–132, Washington, DC, USA, 2012. IEEE Computer Society.

[67] Jason Flinn, SoYoung Park, and M. Satyanarayanan. Balancing performance, energy, and quality in pervasive computing. In *Distributed Computing Systems, 2002. Proceedings. 22nd International Conference on*, pages 217–226, 2002.

[68] Fedor V. Fomin, Daniel Lokshtanov, and Saket Saurabh. An exact algorithm for minimum distortion embedding. In Christophe Paul and Michel Habib, editors, *Graph-Theoretic Concepts in Computer Science*, pages 112–121, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[69] A. Fox, C. Eichelberger, J. Hughes, and S. Lyon. Spatio-temporal indexing in non-relational distributed databases. In *Big Data, 2013 IEEE International Conference on*, pages 291–299, Oct 2013.

[70] Michael J. Franklin, Shawn R. Jeffery, Sailesh Krishnamurthy, and Frederick Reiss. Design considerations for high fan-in systems: The hifi approach. In *In CIDR*, pages 290–304, 2005.

[71] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[72] N. K. Giang, M. Blackstock, R. Lea, and V. C. M. Leung. Developing iot applications in the fog: A distributed dataflow approach. In *2015 5th International Conference on the Internet of Things (IOT)*, pages 155–162, Oct 2015.

[73] P. B. Gibbons, B. Karp, Y. Ke, S. Nath, and Srinivasan Seshan. Irisnet: an architecture for a worldwide sensor web. *IEEE Pervasive Computing*, 2(4):22–33, Oct 2003.

[74] Phillip B. Gibbons and Yossi Matias. New sampling-based summary statistics for improving approximate query answers. In *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*, SIGMOD '98, pages 331–342, New York, NY, USA, 1998. ACM.

[75] Lukasz Golab and M. Tamer Özsu. Issues in data stream management. *SIGMOD Rec.*, 32(2):5–14, June 2003.

[76] Lukasz Golab and M. Tamer Zsu. *Data Stream Management*. Morgan & Claypool Publishers, 2010.

[77] Le Guan, Xu Ke, Meina Song, and Junde Song. A survey of research on mobile cloud computing. In *Computer and Information Science (ICIS), 2011 IEEE/ACIS 10th International Conference on*, may 2011.

[78] Jian Guo, Fangming Liu, Dan Zeng, J.C.S. Lui, and Hai Jin. A cooperative game based allocation for sharing data center networks. In *INFOCOM, 2013 Proceedings IEEE*, pages 2139–2147, April 2013.

[79] Harshit Gupta, Amir Vahid Dastjerdi, Soumya K Ghosh, and Rajkumar Buyya. ifogsim: A toolkit for modeling and simulation of resource management techniques in internet of things, edge and fog computing environments. *arXiv preprint arXiv:1606.02007*, 2016.

[80] Yuri Gurevich, Dirk Leinders, and Jan Van Den Bussche. A theory of stream queries. In *Proceedings of the 11th International Conference on Database Programming Languages*, DBPL'07, pages 153–168, Berlin, Heidelberg, 2007. Springer-Verlag.

[81] K. Ha, P. Pillai, G. Lewis, S. Simanta, S. Clinch, N. Davies, and M. Satyanarayanan. The impact of mobile multimedia applications on data center consolidation. In *2013 IEEE International Conference on Cloud Engineering (IC2E)*, pages 166–176, March 2013.

[82] Kiryong Ha, Padmanabhan Pillai, Wolfgang Richter, Yoshihisa Abe, and Mahadev Satyanarayanan. Just-in-time provisioning for cyber foraging. In *Proceeding of the 11th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '13, pages 153–166, New York, NY, USA, 2013. ACM.

[83] M. A. Hammad, M. F. Mokbel, M. H. Ali, W. G. Aref, A. C. Catlin, A. K. Elmagarmid, M. Eltabakh, M. G. Elfeky, T. M. Ghanem, R. Gwadera, I. F. Ilyas, M. Marzouk, and X. Xiong. Nile: a query processing engine for data streams. In *Proceedings. 20th International Conference on Data Engineering*, pages 851–, March 2004.

[84] Moeen Hassanalieragh, Alex Page, Tolga Soyata, Gaurav Sharma, Mehmet Aktas, Gonzalo Mateos, Burak Kantarci, and Silvana Andreescu. Health monitoring and management using internet-of-things (iot) sensing with cloud-based processing: Opportunities and challenges. In *Proceedings of the 2015 IEEE International Conference on Services Computing*, SCC '15, pages 285–292, Washington, DC, USA, 2015. IEEE Computer Society.

[85] X. Hei, Y. Liu, and K. W. Ross. Iptv over p2p streaming networks: the mesh-pull approach. *IEEE Communications Magazine*, 46(2):86–92, February 2008.

[86] Xiaojun Hei, Chao Liang, Jian Liang, Yong Liu, and K.W. Ross. A measurement study of a large-scale p2p iptv system. *Multimedia, IEEE Transactions on*, 9(8), dec. 2007.

[87] Thomas Heinze, Lars Roediger, Andreas Meister, Yuanzhen Ji, Zbigniew Jerzak, and Christof Fetzer. Online parameter optimization for elastic data stream processing. In *Proceedings of the Sixth ACM Symposium on Cloud Computing*, SoCC '15, pages 276–287, New York, NY, USA, 2015. ACM.

[88] Stijn Heldens, Claudio Martella, Nelly Litvak, and Maarten van Steen. Automated lane detection in crowds using proximity graphs. *CoRR*, abs/1707.01698, 2017.

[89] M. M. Hossain, M. Fotouhi, and R. Hasan. Towards an analysis of security issues, challenges, and open problems in the internet of things. In *2015 IEEE World Congress on Services*, pages 21–28, June 2015.

[90] Ningning Hu, Student Member, Peter Steenkiste, and Senior Member. Evaluation and characterization of available bandwidth probing techniques. *IEEE Journal on Selected Areas in Communications*, 21:879–894, 2003.

[91] J. Huang, G. Liu, Q. Duan, and Y. Yan. Qos-aware service composition for converged network-cloud service provisioning. In *2014 IEEE International Conference on Services Computing*, pages 67–74, June 2014.

[92] Yan Huang and Chengyang Zhang. New data types and operations to support geostreams. In *Proceedings of the 5th International Conference on Geographic Information Science*, GIScience '08, pages 106–118, Berlin, Heidelberg, 2008. Springer-Verlag.

[93] Yuanqiang Huang, Zhongzhi Luan, Rong He, and Depei Qian. Operator placement with qos constraints for distributed stream processing. In *Proceedings of the 7th International Conference on Network and Services Management*, CNSM '11, pages 309–315, Laxenburg, Austria, Austria, 2011. International Federation for Information Processing.

[94] Ryan Huebsch, Joseph M. Hellerstein, Nick Lanham, Boon Thau Loo, Scott Shenker, and Ion Stoica. Querying the internet with pier. In *Proceedings of the 29th International Conference on Very Large Data Bases - Volume 29*, VLDB '03, pages 321–332. VLDB Endowment, 2003.

[95] G. Huerta-Canepa and Dongman Lee. An adaptable application offloading scheme based on application behavior. In *Advanced Information Networking and Applications - Workshops, 2008. AINAW 2008. 22nd International Conference on*, 2008.

[96] L. Berger N. Diner D. Heatley I. Higginbottom L. N. Andersen O. Langeland J. P. Lapierre I. H. McQuinn, D. Reid. Description of the ices hac standard data exchange format. Nov 2005.

[97] Fatemeh Jalali, Olivia J. Smith, Timothy Lynar, and Frank Suits. Cognitive iot gateways: Automatic task sharing and switching between cloud and edge/fog computing. In *Proceedings of the SIGCOMM Posters and Demos*, SIGCOMM Posters and Demos '17, pages 121–123, New York, NY, USA, 2017. ACM.

[98] Y. Jararweh, A. Doulat, O. AlQudah, E. Ahmed, M. Al-Ayyoub, and E. Benkhelifa. The future of mobile cloud computing: Integrating cloudlets and mobile edge computing. In *2016 23rd International Conference on Telecommunications (ICT)*, pages 1–5, May 2016.

[99] Qingchun Jiang and Sharma Chakravarthy. Anatomy of a data stream management system. In *MavHome, Proc. Symposium on Applied Computing*, pages 654–655, 2004.

[100] E. Kehdi and Baochun Li. Incorporating random linear network coding for peer-to-peer network diagnosis. In *INFOCOM, 2010 Proceedings IEEE*, 2010.

[101] Azade Khalaj, Hanan Lutfiyya, and Mark Perry. The proxy-based mobile grid. In Ying Cai, Thomas Magedanz, Minglu Li, Jinchun Xia, and Carlo Giannelli, editors, *Mobile Wireless Middleware, Operating Systems, and Applications*, Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, pages 59–69. Springer Berlin Heidelberg, 2010.

[102] Nikolaos Konstantinou and Dimitrios-Emmanuel Spanos. *Generating Linked Data in Real-time from Sensor Data Streams*, pages 103–126. Springer International Publishing, Cham, 2015.

[103] Balachander Krishnamurthy and Jia Wang. Topology modeling via cluster graphs. In *Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*, 2001.

[104] K. Kumar and Yung-Hsiang Lu. Cloud computing for mobile users: Can offloading computation save energy? *Computer*, 43(4):51 –56, april 2010.

[105] Karthik Kumar, Jibang Liu, Yung-Hsiang Lu, and Bharat Bhargava. A survey of computation offloading for mobile systems. *Mob. Netw. Appl.*, 18(1):129–140, February 2013.

[106] Karthik Kumar, Jibang Liu, Yung-Hsiang Lu, and Bharat Bhargava. A survey of computation offloading for mobile systems. *Mob. Netw. Appl.*, 18(1), February 2013.

[107] Richard Kuntschke and Alfons Kemper. Data stream sharing. Springer Berlin Heidelberg, 2006.

[108] In Lee and Kyoochun Lee. The internet of things (iot): Applications, investments, and challenges for enterprises. 58, 04 2015.

[109] Chris Lesniewski-Laas. A sybil-proof one-hop dht. In *Proceedings of the 1st Workshop on Social Network Systems*, SocialNets '08, pages 19–24, New York, NY, USA, 2008. ACM.

[110] Baochun Li and Di Niu. Random network coding in peer-to-peer networks: From theory to practice. *Proceedings of the IEEE*, 2011.

[111] T. Li, J. Yuan, and M. Torlak. Network throughput optimization for random access narrowband cognitive radio internet of things (nb-cr-iot). *IEEE Internet of Things Journal*, PP(99):1–1, 2018.

[112] Chao Liang and Yong Liu. Vivud: Virtual server cluster based view-upload decoupling for multi-channel p2p video streaming systems. In *Global Telecommunications Conference (GLOBECOM 2010), 2010 IEEE*, 2010.

[113] Bin Liu, Amarnath Gupta, and Ramesh Jain. A live multimedia stream querying system. In *Proceedings of the 2Nd International Workshop on Computer Vision Meets Databases*, CVDB '05, pages 35–42, New York, NY, USA, 2005. ACM.

[114] Fangming Liu, Bo Li, Lili Zhong, Baochun Li, Hai Jin, and Xiaofei Liao. Flash crowd in p2p live streaming systems: Fundamental characteristics and design implications. *Parallel and Distributed Systems, IEEE Transactions on*, 2012.

[115] Qingfeng Liu, Xie Jian, Jicheng Hu, Hongchen Zhao, and Shanshan Zhang. An optimized solution for mobile environment using mobile cloud computing. In *Wireless Communications, Networking and Mobile Computing, 2009. WiCom '09. 5th International Conference on*, sept. 2009.

[116] Yong Liu, Yang Guo, and Chao Liang. A survey on peer-to-peer video streaming systems. *Peer-to-Peer Networking and Applications*, 2 2008.

[117] Samuel Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. Tag: A tiny aggregation service for ad-hoc sensor networks. *SIGOPS Oper. Syst. Rev.*, 36(SI):131–146, December 2002.

[118] Samuel Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. The design of an acquisitional query processor for sensor networks. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, SIGMOD '03, pages 491–502, New York, NY, USA, 2003. ACM.

[119] Redowan Mahmud, Fernando Luiz Koch, and Rajkumar Buyya. Cloud-fog interoperability in iot-enabled healthcare solutions. In *Proceedings of the 19th International Conference on Distributed Computing and Networking*, ICDCN '18, New York, NY, USA, 2018. ACM.

[120] David Maier, Jin Li, Peter Tucker, Kristin Tufte, and Vassilis Papadimos. Semantics of data streams and operators. In *Proceedings of the 10th International Conference on Database Theory*, ICDT'05, pages 37–52, Berlin, Heidelberg, 2005. Springer-Verlag.

[121] M. Maier, M. Chowdhury, B. P. Rimal, and D. P. Van. The tactile internet: vision, recent progress, and open challenges. *IEEE Communications Magazine*, 54(5):138–145, May 2016.

[122] F. Malandrino, C. Casetti, C. Chiasserini, and M. Fiore. Offloading cellular networks through its content download. In *2012 9th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON)*, pages 263–271, June 2012.

[123] C. Miers, M. Simpli andcio, D. Gallo, T. Carvalho, G. Bressan, V. Souza, P. Karlsson, and A. Damola. I2ts01 - a taxonomy for locality algorithms on peer-to-peer networks. *Latin America Transactions, IEEE*, 2010.

[124] Jeremiah Miller, Miles Raymond, Josh Archer, Seid Adem, Leo Hansel, Sushma Konda, Malik Luti, Yao Zhao, Ankur Teredesai, and Mohamed Ali. An extensibility approach for spatio-temporal stream processing using microsoft streaminsight. In *Proceedings of the 12th International Conference on Advances in Spatial and Temporal Databases*, SSTD'11, pages 496–501, Berlin, Heidelberg, 2011. Springer-Verlag.

[125] A. Munir, P. Kansakar, and S. U. Khan. Ifciot: Integrated fog cloud iot: A novel architectural paradigm for the future internet of things. *IEEE Consumer Electronics Magazine*, 6(3):74–82, July 2017.

[126] R. N. Murty, G. Mainland, I. Rose, A. R. Chowdhury, A. Gosain, J. Bers, and M. Welsh. Citysense: An urban-scale wireless sensor network and testbed. In *Technologies for Homeland Security, 2008 IEEE Conference on*, pages 583–588, May 2008.

[127] N. Mkitalo, A. Ometov, J. Kannisto, S. Andreev, Y. Koucheryavy, and T. Mikkonen. Safe and secure execution at the network edge: A framework for coordinating cloud, fog, and edge. *IEEE Software*, pages 1–1, 2018.

[128] M. Nardelli. Framework for data stream applications in a distributed cloud. ZEUS Workshop, 2016.

[129] Suman Nath, Jie Liu, and Feng Zhao. Sensormap for wide-area sensor webs. *Computer*, 40(7):90–93, July 2007.

[130] L. Neumeyer, B. Robbins, A. Nair, and A. Kesari. S4: Distributed stream computing platform. In *2010 IEEE International Conference on Data Mining Workshops*, pages 170–177, Dec 2010.

[131] Anh Tuan Nguyen, Baochun Li, and F. Eliassen. Chameleon: Adaptive peer-to-peer streaming with network coding. In *INFOCOM, 2010 Proceedings IEEE*, 2010.

[132] Kien Nguyen, Thinh Nguyen, and Sen ching Cheung. Peer-to-peer streaming with hierarchical network coding. In *Multimedia and Expo, 2007 IEEE International Conference on*, 2007.

[133] Silvia Nittel. Real-time sensor data streams. *SIGSPATIAL Special*, 7(2):22–28, September 2015.

[134] L. O'Callaghan, N. Mishra, A. Meyerson, S. Guha, and R. Motwani. Streaming-data algorithms for high-quality clustering. In *Data Engineering, 2002. Proceedings. 18th International Conference on*, pages 685–694, 2002.

[135] D. O'Keeffe, T. Salonidis, and P. Pietzuch. Network-aware stream query processing in mobile ad-hoc networks. In *MILCOM 2015 - 2015 IEEE Military Communications Conference*, pages 1335–1340, Oct 2015.

[136] P. Pietzuch, J. Ledlie, J. Shneidman, M. Roussopoulos, M. Welsh, and M. Seltzer. Network-aware operator placement for stream-processing systems. In *22nd International Conference on Data Engineering (ICDE'06)*, pages 49–49, April 2006.

[137] E. J. Qaisar. Introduction to cloud computing for developers: Key concepts, the players and their offerings. In *2012 IEEE TCF Information Technology Professional Conference*, pages 1–6, March 2012.

[138] Alfonso Ariza Quintana, Eduardo Casilari, and Alicia Triviño. Implementation of manet routing protocols on omnet++, 2008.

[139] Moo-Ryong Ra, Anmol Sheth, Lily Mummert, Padmanabhan Pillai, David Wetherall, and Ramesh Govindan. Odessa: enabling interactive perception applications on mobile devices. In *Proceedings of the 9th international conference on Mobile systems, applications, and services*, MobiSys '11, pages 43–56, 2011.

[140] W. Raab, N. Bruels, U. Hachmann, J. Harnisch, U. Ramacher, C. Sauer, and A. Techmer. A 100-gops programmable processor for vehicle vision systems. *IEEE Design Test of Computers*, 20(1):8–15, Jan 2003.

[141] Venkatesh Raghavan, Yali Zhu, Elke A. Rundensteiner, and Daniel Dougherty. Multi-join continuous query optimization: Covering the spectrum of linear, acyclic, and cyclic queries. In *Proceedings of the 26th British National Conference on Databases: Dataspace: The Final Frontier*, BNCOD 26, pages 91–106, Berlin, Heidelberg, 2009. Springer-Verlag.

[142] K.K. Ramachandran and B. Sikdar. An analytic framework for modeling peer to peer networks. In *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, volume 3, pages 215–269 vol. 3, 2005.

[143] Aravinda S. Rao, Jayavardhana Gubbi, Slaven Marusic, and Marimuthu Palaniswami. Estimation of crowd density by clustering motion cues. *Vis. Comput.*, 31(11):1533–1552, November 2015.

[144] S. Rizou, F. Durr, and K. Rothermel. Solving the multi-operator placement problem in large-scale operator networks. In *2010 Proceedings of 19th International Conference on Computer Communications and Networks*, pages 1–6, Aug 2010.

[145] B. Rochwerger, D. Breitgand, A. Epstein, D. Hadas, I. Loy, K. Nagin, J. Tordsson, C. Ragusa, M. Villari, S. Clayman, E. Levy, A. Maraschini, P. Massonet, H. Muoz, and G. Tofetti. Reservoir - when one cloud is not enough. *Computer*, 44(3):44 –51, march 2011.

[146] P.D. Rodrigues, C. Ribeiro, and L. Veiga. Incentive mechanisms in peer-to-peer networks. In *Parallel Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on*, april 2010.

[147] Philippe Lamarre Yves Caniou Roland Kotto-Kombi, Nicolas Lumineau. Parallel and dis- tributed stream processing: Systems classi cation and speci c issues. 2015.

[148] I. J. Rudas. Cloud computing in education. In *2012 IEEE 10th International Conference on Emerging eLearning Technologies and Applications (ICETA)*, pages 327–327, Nov 2012.

[149] J. Rckert, B. Richerzhagen, E. Lidanski, R. Steinmetz, and D. Hausheer. Topt: Supporting flash crowd events in hybrid overlay-based live streaming. In *2015 IFIP Networking Conference (IFIP Networking)*, pages 1–9, May 2015.

[150] L. Sanchez, J. A. Galache, V. Gutierrez, J. M. Hernandez, J. Bernat, A. Gluhak, and T. Garcia. Smartsantander: The meeting point between future internet research and experimentation and the smart cities. In *Future Network Mobile Summit (FutureNetw), 2011*, pages 1–8, June 2011.

[151] João Santos, Joel J.P.C. Rodrigues, Bruno M.C. Silva, João Casal, Kashif Saleem, and Victor Denisov. An iot-based mobile gateway for intelligent personal assistants on mobile health environments. *J. Netw. Comput. Appl.*, 71(C):194–204, August 2016.

[152] M. Satyanarayanan. The emergence of edge computing. *Computer*, 50(1):30–39, Jan 2017.

[153] M. Satyanarayanan, P. Simoens, Y. Xiao, P. Pillai, Z. Chen, K. Ha, W. Hu, and B. Amos. Edge analytics in the internet of things. *IEEE Pervasive Computing*, 14(2):24–31, Apr 2015.

[154] Mahadev Satyanarayanan, P. Bahl, R. Caceres, and N. Davies. The case for vm-based cloudlets in mobile computing. *Pervasive Computing, IEEE*, 2009.

[155] B. Satzger, W. Hummer, P. Leitner, and S. Dustdar. Esc: Towards an elastic stream computing platform for the cloud. In *2011 IEEE 4th International Conference on Cloud Computing*, pages 348–355, July 2011.

[156] P. Seeling and M. Reisslein. Video transport evaluation with H.264 video traces. *IEEE Communications Surveys and Tutorials, in print*, 2012. Traces available at `trace.eas.asu.edu`.

[157] S. A. Shaikh, Y. Watanabe, Y. Wang, and H. Kitagawa. Smart query execution for event-driven stream processing. In *2016 IEEE Second International Conference on Multimedia Big Data (BigMM)*, pages 97–104, April 2016.

[158] Mohsen Sharifi, Somayeh Kafaie, and Omid Kashefi. A survey and taxonomy of cyber foraging of mobile devices. *Communications Surveys Tutorials, IEEE*, 14(4):1232–1243, 2012.

[159] Ahmed Shawish and Maria Salama. *Cloud Computing: Paradigms and Technologies*, pages 39–67. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014.

[160] Chao Shen, Yan Huang, and Jason W. Powell. The design of a benchmark for geo-stream management systems. In *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, GIS '11, pages 409–412, New York, NY, USA, 2011. ACM.

[161] T. Silverston, O. Fourmaux, K. Salamatian, and K. Cho. On fairness and locality in p2p-tv through large-scale measurement experiment. In *Global Telecommunications Conference (GLOBECOM 2010), 2010 IEEE*, 2010.

[162] Pieter Simoens, Yu Xiao, Padmanabhan Pillai, Zhuo Chen, Kiryong Ha, and Mahadev Satyanarayanan. Scalable crowd-sourcing of video from mobile devices. In *Proceeding of the 11th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '13, pages 139–152, New York, NY, USA, 2013. ACM.

[163] Vasilios A. Siris and Dimitrios Kalyvas. Enhancing mobile data offloading with mobility prediction and prefetching. In *Proceedings of the Seventh ACM International Workshop on Mobility in the Evolving Internet Architecture*, MobiArch '12, pages 17–22, New York, NY, USA, 2012. ACM.

[164] Weiguang Song and Xiaolong Su. Review of mobile cloud computing. In *Communication Software and Networks (ICCSN), 2011 IEEE 3rd International Conference on*, may 2011.

[165] Tolga Soyata, Rajani Muraleedharan, Jonathan Langdon, Colin Funai, Scott Ames, Minseok Kwon, and Wendi Heinzelman. Combat: mobile-cloud-based compute/communications infrastructure for battlefield applications. In *SPIE defense, security, and sensing*, pages 84030K–84030K. International Society for Optics and Photonics, 2012.

[166] Salvatore Spoto, Rossano Gaeta, Marco Grangetto, and Matteo Sereno. Analysis of pplive through active and passive measurements. In *Proceedings of the 2009 IEEE International Symposium on Parallel&Distributed Processing*, Washington, DC, USA, 2009. IEEE Computer Society.

[167] Utkarsh Srivastava, Kamesh Munagala, and Jennifer Widom. Operator placement for in-network stream query processing. In *Proceedings of the Twenty-fourth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '05, pages 250–258, New York, NY, USA, 2005. ACM.

[168] I. Stanoi, G. Mihaila, T. Palpanas, and C. Lang. Whitewater: Distributed processing of fast streams. *IEEE Transactions on Knowledge and Data Engineering*, 19(9):1214–1226, Sept 2007.

[169] Till Steinbach, Hermand Dieumo Kenfack, Franz Korf, and Thomas C. Schmidt. An Extension of the OMNeT++ INET Framework for Simulating Real-time Ethernet with High Accuracy. In *SIMUTools*, pages 375–382, USA, 2011.

[170] I. Stoica, R. Morris, D. Liben-Nowell, D.R. Karger, M.F. Kaashoek, F. Dabek, and H. Balakrishnan. Chord: a scalable peer-to-peer lookup protocol for internet applications. *Networking, IEEE/ACM Transactions on*, 2003.

[171] Michael Stonebraker, Uğur Çetintemel, and Stan Zdonik. The 8 requirements of real-time stream processing. *SIGMOD Rec.*, 34(4):42–47, December 2005.

[172] Ya-Yunn Su and Jason Flinn. Slingshot: deploying stateful services in wireless hotspots. In *Proceedings of the 3rd international conference on Mobile systems, applications, and services*, MobiSys '05, pages 79–92, 2005.

[173] Mark Sullivan and Andrew Heybey. Tribeca: A system for managing large databases of network traffic. In *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, ATEC '98, pages 2–2, Berkeley, CA, USA, 1998. USENIX Association.

[174] M. Taneja and A. Davy. Resource aware placement of iot application modules in fog-cloud computing paradigm. In *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pages 1222–1228, May 2017.

[175] C. Thoma, A. Labrinidis, and A. J. Lee. Automated operator placement in distributed data stream management systems subject to user constraints. In *2014 IEEE 30th International Conference on Data Engineering Workshops*, pages 310–316, March 2014.

[176] M. Tsai and Y. Lin. How the usage of mobile multimedia internet devices changes internet tv consumer behaviors in taiwan: Using pps.tv (ppstream) as an example. In *2015 Portland International Conference on Management of Engineering and Technology (PICMET)*, pages 1467–1476, Aug 2015.

[177] Peter A. Tucker, David Maier, Tim Sheard, and Leonidas Fegaras. Exploiting punctuation semantics in continuous data streams. *IEEE Trans. on Knowl. and Data Eng.*, 15(3):555–568, March 2003.

[178] Blesson Varghese and Rajkumar Buyya. Next generation cloud computing: New trends and research directions. *CoRR*, abs/1707.07452, 2017.

[179] Neelam Bhardwaj Verma, Manisha and Arun Kumar Yadav. Real time efficient scheduling algorithm for load balancing in fog computing environment. Information Technology and Computer Science,, 2016.

[180] A.S. Wagner. Embedding trees into the hypercube. October 1987.

[181] M. Wang, L. Xu, and B. Ramamurthy. Comparing multi-channel peer-to-peer video streaming system designs. In *2010 17th IEEE Workshop on Local Metropolitan Area Networks (LANMAN)*, pages 1–6, May 2010.

[182] Mea Wang and Baochun Li. Lava: A reality check of network coding in peer-to-peer live streaming. In *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*, 2007.

[183] Mea Wang and Baochun Li. R2: Random push with random network coding in live peer-to-peer streaming. *Selected Areas in Communications, IEEE Journal on*, 2007.

[184] Miao Wang, Lisong Xu, and B. Ramamurthy. Comparing multi-channel peer-to-peer video streaming system designs. In *Local and Metropolitan Area Networks (LANMAN), 2010 17th IEEE Workshop on*, 2010.

[185] M. Wichtlhuber, B. Richerzhagen, J. Rckert, and D. Hausheer. Transit: Supporting transitions in peer-to-peer live video streaming. In *2014 IFIP Networking Conference*, June 2014.

[186] Chuan Wu, Baochun Li, and Shuqiao Zhao. Multi-channel live p2p streaming: Refocusing on servers. In *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE*, 2008.

[187] Di Wu, Chao Liang, Yong Liu, and K. Ross. View-upload decoupling: A redesign of multi-channel p2p video systems. In *INFOCOM 2009, IEEE*, 2009.

[188] Di Wu, Yong Liu, and K.W. Ross. Modeling and analysis of multichannel p2p live video systems. *Networking, IEEE/ACM Transactions on*, 2010.

[189] Haibo Wu, Hai Jiang, Jing Liu, Yi Sun, Jun Li, and Zhongcheng Li. How p2p live streaming systems scale quickly under a flash crowd? In *Performance Computing and Communications Conference (IPCCC), 2011 IEEE 30th International*, 2011.

[190] J. Xu, Z. Chen, J. Tang, and S. Su. T-storm: Traffic-aware online scheduling in storm. In *2014 IEEE 34th International Conference on Distributed Computing Systems*, pages 535–544, June 2014.

[191] Jielong Xu, Zhenhua Chen, Jian Tang, and Sen Su. T-storm: Traffic-aware online scheduling in storm. In *Proceedings of the 2014 IEEE 34th International Conference on Distributed Computing Systems*, ICDCS '14, pages 535–544, Washington, DC, USA, 2014. IEEE Computer Society.

[192] Jun Yang, Dai Bin, Benxiong Huang, and Huang Chen. Topology awareness of hierarchical peer-to-peer network based on network coding with feedback. In *Intelligent Systems and Applications (ISA), 2010 2nd International Workshop on*, 2010.

[193] Lei Yang, Jiannong Cao, Shaojie Tang, Tao Li, and A.T.S Chan. A framework for partitioning and execution of data stream applications in mobile cloud computing. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pages 794–802, 2012.

[194] Lei Yang, Jiannong Cao, Yin Yuan, Tao Li, Andy Han, and Alvin Chan. A framework for partitioning and execution of data stream applications in mobile cloud computing. *SIGMETRICS Perform. Eval. Rev.*, 40(4):23–32, April 2013.

[195] Wujian Ye and Kyungsan Cho. Hybrid p2p traffic classification with heuristic rules and machine learning. *Soft Computing*, 18(9):1815–1827, Sep 2014.

[196] Shui Yu and Zhongwen Li. Massive data delivery in unstructured peer-to-peer networks with network coding. In *Computer and Information Science, 2007. ICIS 2007. 6th IEEE/ACIS International Conference on*, 2007.

[197] H. Yue, L. Guo, R. Li, H. Asaeda, and Y. Fang. Dataclouds: Enabling community-based data-centric services over the internet of things. *IEEE Internet of Things Journal*, 1(5):472–482, Oct 2014.

[198] D. Zhang, L. T. Yang, and H. Huang. Searching in internet of things: Vision and challenges. In *2011 IEEE Ninth International Symposium on Parallel and Distributed Processing with Applications*, pages 201–206, May 2011.

[199] Kan Zhang, N. Antonopoulos, and Zaigham Mahmood. A review of incentive mechanism in peer-to-peer systems. In *Advances in P2P Systems, 2009. AP2PS '09. First International Conference on*, 2009.

[200] Xinyan Zhang, Jiangchuan Liu, Bo Li, and Y.-S.P. Yum. Coolstreaming/donet: a data-driven overlay network for peer-to-peer live media streaming. In *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, 2005.

[201] Jianlong Zhong and Bingsheng He. Medusa: A parallel graph processing system on graphics processors. *SIGMOD Rec.*, 43(2):35–40, December 2014.

[202] Yongluan Zhou, Beng Chin Ooi, Kian-Lee Tan, and Ji Wu. Efficient dynamic operator placement in a locally distributed continuous query system. In *Proceedings of the 2006 Confederated International Conference on On the Move to Meaningful Internet Systems: CoopIS, DOA, GADA, and ODBASE - Volume Part I*, ODBASE'06/OTM'06, pages 54–71, Berlin, Heidelberg, 2006. Springer-Verlag.

Appendix A

Simulator

A.1 Introduction to the Simulator

For the simulation, iFogSim [79] was extended. iFogsim uses the event simulation functionalities of CloudSim [33]. Basically, the core CloudSim layer is responsible for handling events between fog computing entities in iFogSim. The main components of this simulation are the following:

- **Fog Node:** In iFogSim [79] the FogDevice class was used for a fog node. FogDevice class specifies hardware characteristics of a fog node and its connections to other fog nodes and sensors. FogDevice class is an extension of the PowerDatacenter class in CloudSim [33]. The major attributes of the FogDevice class are memory, processor, storage size, uplink and downlink bandwidths. In the simulation by using *createFogDevice* function we can create a fog node and set up the required specifications. The required specifications for a fog node are: $< node-name; MIPS; Memory; upload-bandwidth; download-bandwidth >$.

- **Sensor:** This class contains attributes representing the characteristics of a sensor such as a sensor's connectivity and output attributes. This class includes an attribute which specifies a gateway to a fog device. Moreover, this class defines the output characteristics of the sensor. There is no processing capacity was considered for sensors in this simulation.

- **Tuple:** Tuples are represented as instances of the Tuple class in iFogSim. iFogSim inherited Tuple class from the Cloudlet class of CloudSim. A tuple in CloudSim is characterized by its type and the source and destination application component. In this simulation the same specifications were used as described by Verma et al [179]. The attributes of the class specify the processing speed requirements (defined in terms of Million Instructions Per Second).

- **Application:** An application is modeled as a directed acyclic graph (DAG), the vertices of the DAG representing modules that perform processing on incoming data (tuple) and edges denoting data-dependencies between modules.

iFogSim/CloudSim does not have any built-in classes for simulating the physical network. This is a problem for experimentation where we want to assess the impact of the distance

between the data sources and some sort of processing entity. This requires that we modify iFogSim so that the time that the data transmission time is calculated. For this work we developed a new class, Gateway, that is responsible for handling the incoming traffic to a processing entity (fog node or cloud). The Gateway is responsible for calculating delay for incoming tuples.

A.1.1 Data Stream Processing in the Simulator

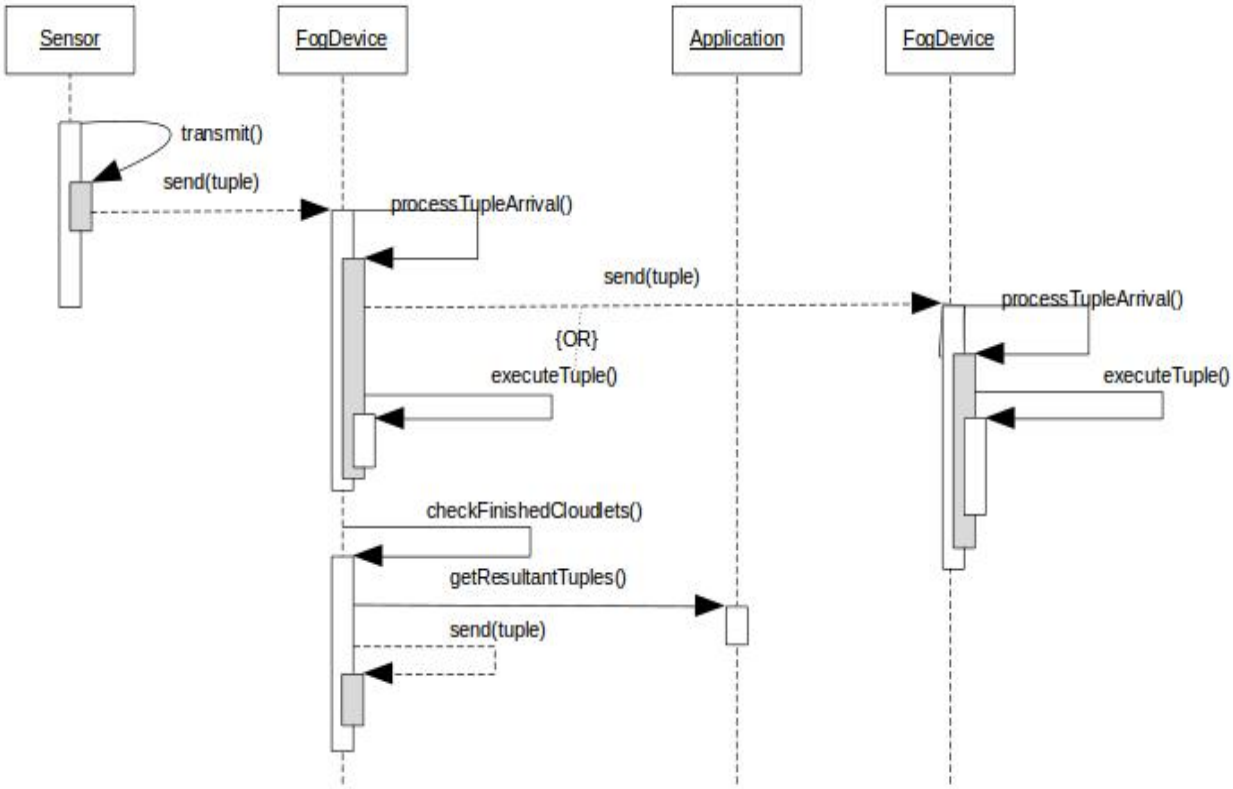

Figure A.1: Sequence diagram of the generation and execution [79]

A sequence diagram demonstrating tuple emission and subsequent execution is shown in Fig. A.1 [79]. In the simulator the processing on a data stream is performed as follows:

- A tuple is generated by a sensor and sent to a fog node or a cloud. The function *send*() is used to send the tuple to another entity.
- Once the tuple reaches the fog node or a cloud the *processTupleArrival*() function is called to handle an incoming tuple. Also, the latency for the received tuple is calculated. Latency calculation is described in Section A.2. In case the tuple needs to be routed to another Fog device, it is sent immediately without processing.

- The *executeTuple*() function is called to process the arrived tuple. The *executeTuple*() function in the processing node contains the tuple processing logic where the device updates its resource utilization. The *executeTuple*() method in the processing node class contains the tuple processing logic. The function *checkCloudletCompletion*() is called on the Fog device on completion of execution of the tuple.

A.2 iFogSim Extension

This section describes how network delay is introduced in the simulation and how end-to-end delay is measured in the simulation. iFogSim [79] only uses a constant delay for each connection between any two nodes, however we added a feature to dynamically calculate the network latency as the network traffic changes. This section has two parts: (1) Setting up the delay in the Simulator; (2) Calculation of End-to-End delay metric.

A.2.1 Setting up Delay in the iFogSim

We added module to iFogSim which is used to simulate a network of routers and switches. This model maintains an $n \times n$ matrix where n represents the number of routers or switches. Fig. A.2 illustrates an example of the underlay network related to the adjacency matrix in Fig. A.3.

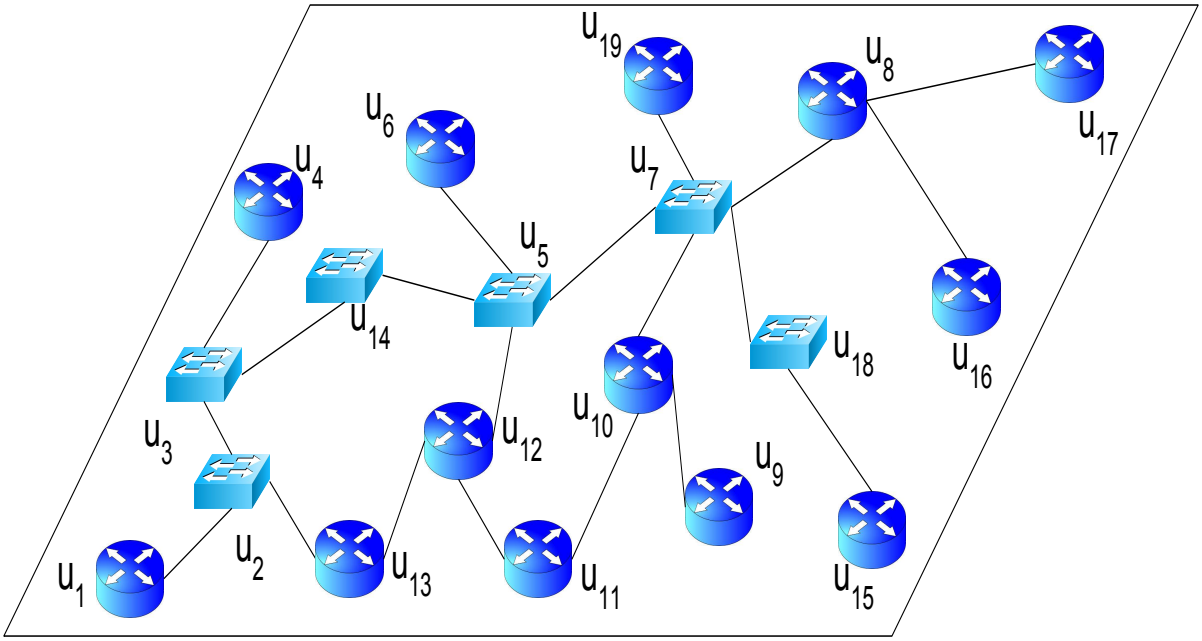

Figure A.2: An Example of an Underlay Network

A transmission rate matrix corresponding to the adjacency matrix was introduced to specify the weight (packet transmission rate) for the underlay adjacency matrix. Each entry in a

A=

	u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8	u_9	u_{10}	u_{11}	u_{12}	u_{13}	u_{14}	u_{15}	u_{16}	u_{17}	u_{18}	u_{19}
u_1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
u_2	1	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
u_3	0	1	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
u_4	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
u_5	0	0	0	0	0	1	1	0	0	0	0	1	0	1	0	0	0	0	0
u_6	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
u_7	0	0	0	0	1	0	0	1	0	1	0	0	0	0	0	0	0	1	1
u_8	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	0	0
u_9	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
u_{10}	0	0	0	0	0	0	1	0	1	0	1	0	0	0	0	0	0	0	0
u_{11}	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0
u_{12}	0	0	0	0	1	0	0	0	0	0	1	0	1	0	0	0	0	0	0
u_{13}	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
u_{14}	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
u_{15}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
u_{16}	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
u_{17}	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
u_{18}	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0
u_{19}	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0

Figure A.3: An Example of an Underlay Adjecency Martrix

transmission rate matrix represents a transmission rate in a link between two adjacent nodes (such as routers).

End-to-End Calculation

In a communication network the amount of time needed for a packet to be transferred between two nodes is known as end-to-end delay. As Fig .A.5 shows the end-to-end delay consists of following parts: (1) Processing delay; (2) Queuing delay; (3) Propagation delay; (4) Transmission delay.

$$D_{End-to-End} = D_{processing} + D_{propagation} + D_{transmission} + D_{queue} \tag{A.1}$$

The following explain the calculation of the end-to-end delay:

- **Processing Delay:** Processing delay is the time it takes routers to process the packet header. The processing delay is not taken into consideration in our extension of iFogSim.
- **Propagation Delay:** Propagation delay is defined as the amount of time for a signal to reach its destination (i.e., the amount of time it takes for a bit to propagate from one router to the next). The propagation delay is not taken into consideration in our extension of iFogSim.

	u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8	u_9	u_{10}	u_{11}	u_{12}	u_{13}	u_{14}	u_{15}	u_{16}	u_{17}	u_{18}
u_1		$r_{1,2}$																
u_2	$r_{1,2}$		$r_{2,3}$										$r_{2,13}$					
u_3		$r_{2,3}$		$r_{3,4}$										$r_{3,14}$				
u_4			$r_{3,4}$															
u_5						$r_{6,5}$	$r_{7,5}$					$r_{12,5}$		$r_{14,5}$				
u_6					$r_{6,5}$													
u_7					$r_{7,5}$			$r_{7,8}$		$r_{10,7}$								$r_{7,18}$
u_8							$r_{7,8}$									$r_{16,8}$	$r_{17,8}$	
u_9										$r_{10,9}$								
u_{10}							$r_{10,7}$		$r_{10,9}$		$r_{10,11}$							
u_{11}										$r_{10,11}$		$r_{12,11}$						
u_{12}					$r_{12,5}$						$r_{12,11}$		$r_{12,13}$					
u_{13}		$r_{2,13}$										$r_{12,13}$						
u_{14}			$r_{3,14}$		$r_{14,5}$													
u_{15}																		$r_{15,18}$
u_{16}								$r_{16,8}$										
u_{17}								$r_{17,8}$										
u_{18}							$r_{7,18}$								$r_{15,18}$			

Figure A.4: Example of a Transmission Rate Matrix for an Underlay Network

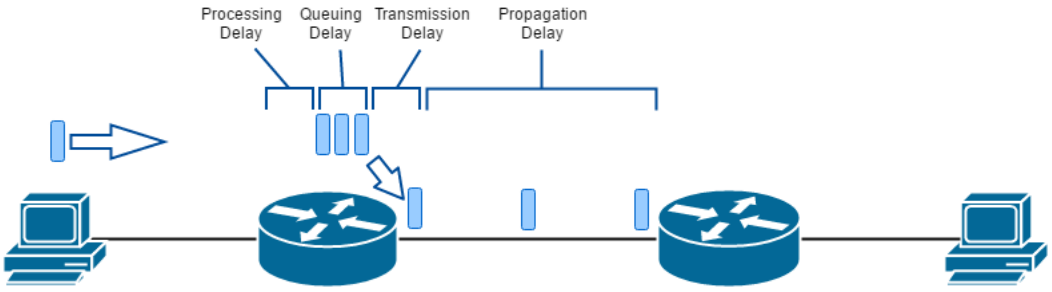

Figure A.5: An Example of an End-to-End Delay [20]

- **Transmission Delay :** $D_{transmission}$ is obtained from the rate adjacency matrix. Transmission delay is defined as the amount of time required for the router to push out a packet. A packet transmission time in seconds can be obtained from the packet size in bit and the transmission rate or bit rate in $\frac{bit}{s}$ as: $\frac{packetSize}{bitRate}$.

- **Queuing delay:** Queuing delay is the time the packet spends in a router's queue. Queuing delay represents how long would a packet is kept in a router queue.

A.2.2 Gateway

The *iFogSim*/*CloudSim* has no actual entities for simulating network entities, such as routers or switches. Therefore, this simulation uses the following method to show the

affect of the queuing network related to the routers or switches in the underlay network. Each fog node/cloud in the simulation uses an instance of a module called "Gateway". Figure A.7 is used to illustrate the Gateway. As seen in Fig. A.7 all data sources send generated data stream to the processing node. However, in the simulation as it was illustrated in Fig. A.8, all the data streams from all the data sources are passed though the gateway (each fog node has a gateway module).

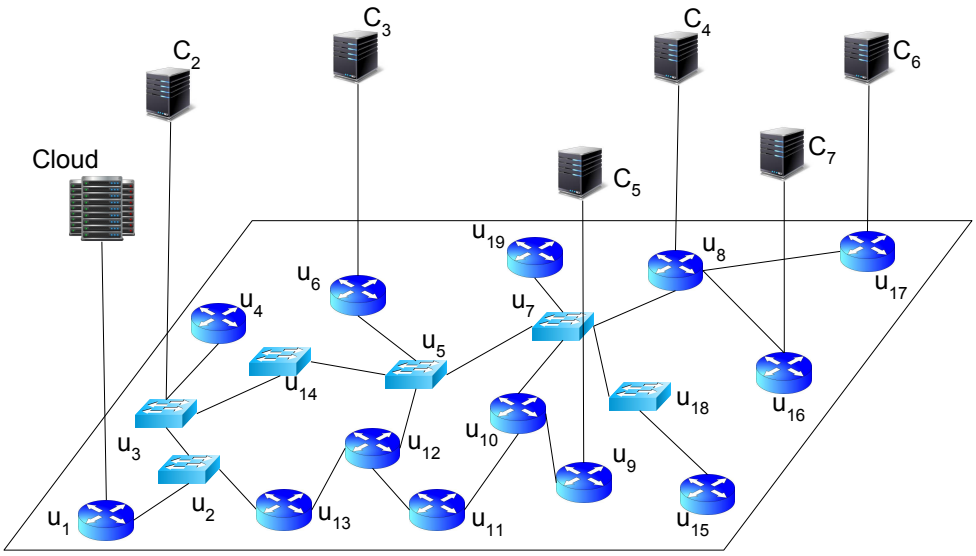

Figure A.6: An Example of a Cloud and Fog Nodes

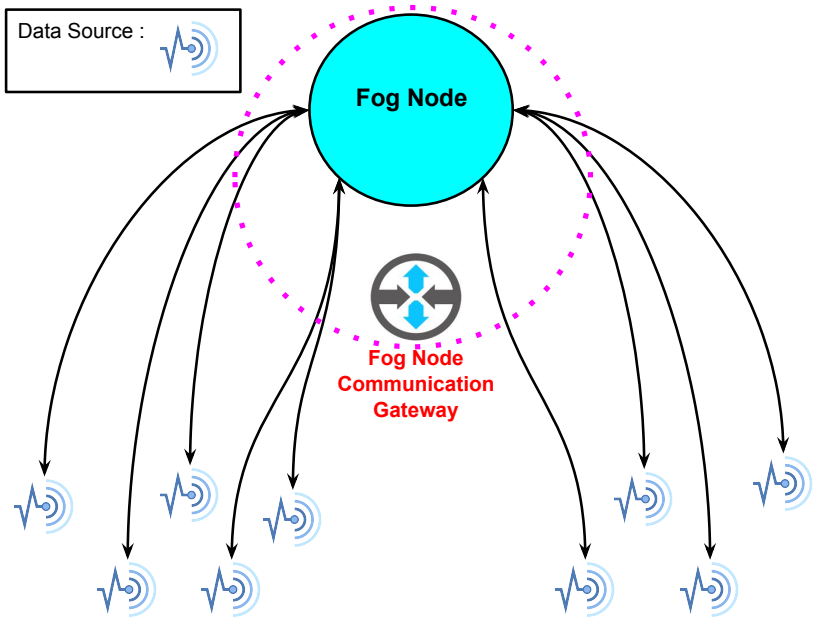

Figure A.7: An Example of Fog Node and Data Sources

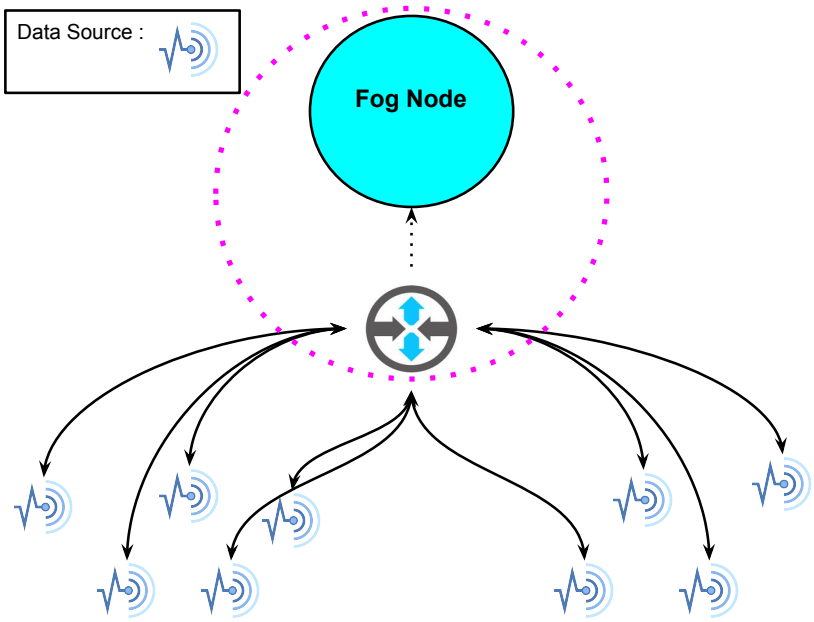

Figure A.8: An Example of Fog Node and Data Sources with a Gateway

Gateway: A Gateway is responsible for calculating and adding the queuing delay for incoming traffic to a fog node/cloud. Considering an entity e_i in the simulation. An entity can be a sensor, a fog node, and a cloud. A gateway of an entity such as e_i calculates the queuing delay by using gateway rate between e_i and all the sender entities to e_i. Equ. A.2 is used to calculate the gateway rate between e_i and e_j when e_j sends a packet of length L to e_i.

$$\frac{L}{R_{Gateway}} = \frac{L}{r_1} + \frac{L}{r_2} + ... + \frac{L}{r_i} \tag{A.2}$$

where r_i represents the transmission rate of a router i belong to a shortest path from e_j to e_i. Equation A.2 can be simplified at Equation A.3.

$$\frac{1}{R_{Gateway}} = \frac{1}{r_1} + \frac{1}{r_2} + ... + \frac{1}{r_i} \tag{A.3}$$

Consider the example in Fig A.9(a). The entity X sends tuples to the entity Y. The gateway associated with entity Y receives all the tuples from X. The gateway associated with Y is responsible for adding queuing time to the received tuples before placing them in a buffer associated with entity Y. The gateway associated with Y uses an adjacency matrix to obtain the path from X to Y. As an example the path from X to Y is illustrated in Fig A.9(b). As Fig A.9(b) illustrates, it is required for the entity X to use a path which includes routers with the following transmission rate: r_1, r_2, r_3, and r_4 to send a packet to the fog node Y. The path from X to Y is obtained from adjacency matrix which was introduced in Section A.2.1. Let us assume that X sends a packet of length L to the entity Y. The transmission rate of a Gateway associated with Y is obtained from equation A.4.

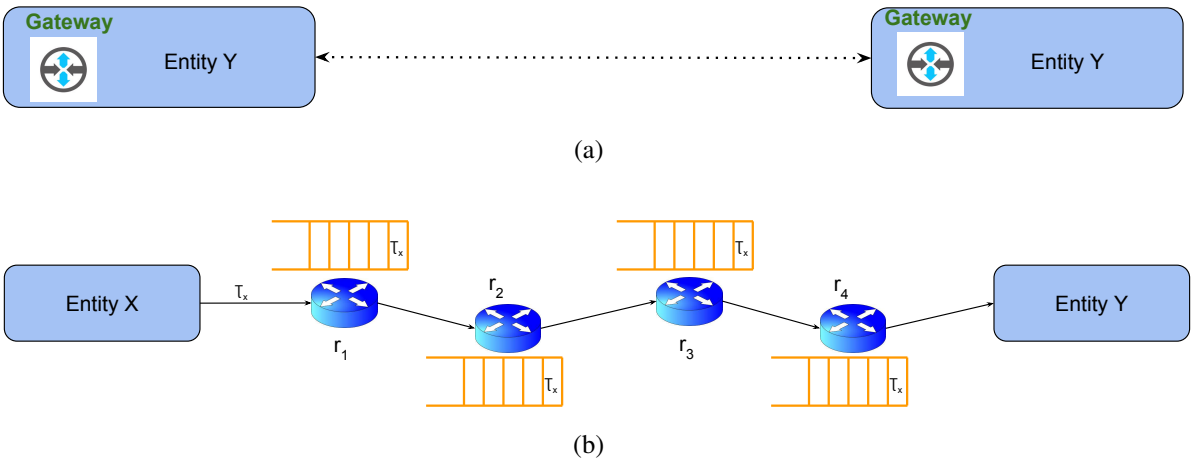

Figure A.9: (a) An Example of two entities in the Simulation, (b) An Example of Two entities with Physical Network Connection

$$\frac{1}{R_Y} = \frac{1}{r_1} + \frac{1}{r_2} + \frac{1}{r_3} + \frac{1}{r_4} \tag{A.4}$$

Equation A.5 calculates the transmission rate for the Gateway in Fig A.9(b) by solving the equation A.4 for $R_{Gateway}$.

$$R_Y = \frac{r_1 \times r_2 \times r_3 \times r_4}{(r_2 \times r_3 \times r_4) + (r_1 \times r_3 \times r_4) + (r_1 \times r_2 \times r_4) + (r_1 \times r_2 \times r_3)} \tag{A.5}$$

Equ A.5 is used for calculating the rate gateway rate between any two fog nodes X and Y.

Now, consider an example that was illustrated in Fog A.10 where entities X, Z, W, V, and U sends tuples to an entity Y. The gateway associated with entity Y requires to calculate the transmission rate for each of X, Z, W, V, and U. In this example the gateway in Y requires to calculate the transmission rate from each X, Z, W, V, and U. In the other words, Y uses R_X, R_Z, R_W, R_V, and R_U to calculated the queuing time form X, Z, W, V, and U. Entity Y can obtain the shortest path from adjacency matrix. Fig. A.11 shows the example after Y obtains the shortest path to X, Z, W, V, and U from the adjacency matrix.

Fig A.11 an example where there are more than one entity are connected to a entity. For example, $R_X = \frac{r_7 \times r_8 \times r_9 \times r_{10}}{r_8 \times r_9 \times r_{10} + r_7 \times r_9 \times r_{10} + r_7 \times r_8 \times r_{10} + r_7 \times r_8 \times r_9}$ where R_X represents the transmission rate that the gateway associated with Y uses to calculate the queuing time for tuples from X. In the same way R_w, R_U, R_Z, and R_V will be calculated.

There is a queue associated with a gateway. Received tuples from senders are placed in the gateway queue first based on their arrival time-stamp. Now let's assume that Fig **??** illustrated a snap of the Gateway's queue in Fig A.11 and τ_j represents a sent tuple from entity j. Let's start from the beginning of the queue. Because τ_X is in the front of the queue and there is no other tuple in front of it therefore the Gateway dose not add any queuing time to τ_X. Let's considering τ_Z in the queue. τ_Z and τ_X passed thought the same underlay path as it was illustrated in Fig

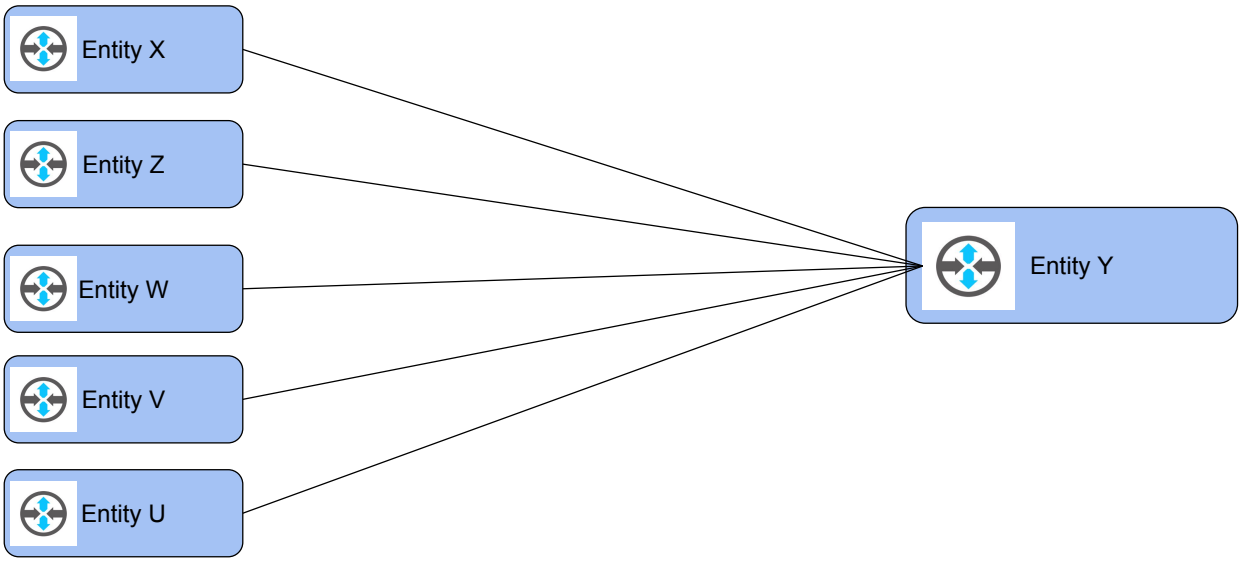

Figure A.10: An Example of Five Fog Nodes in the Simulation

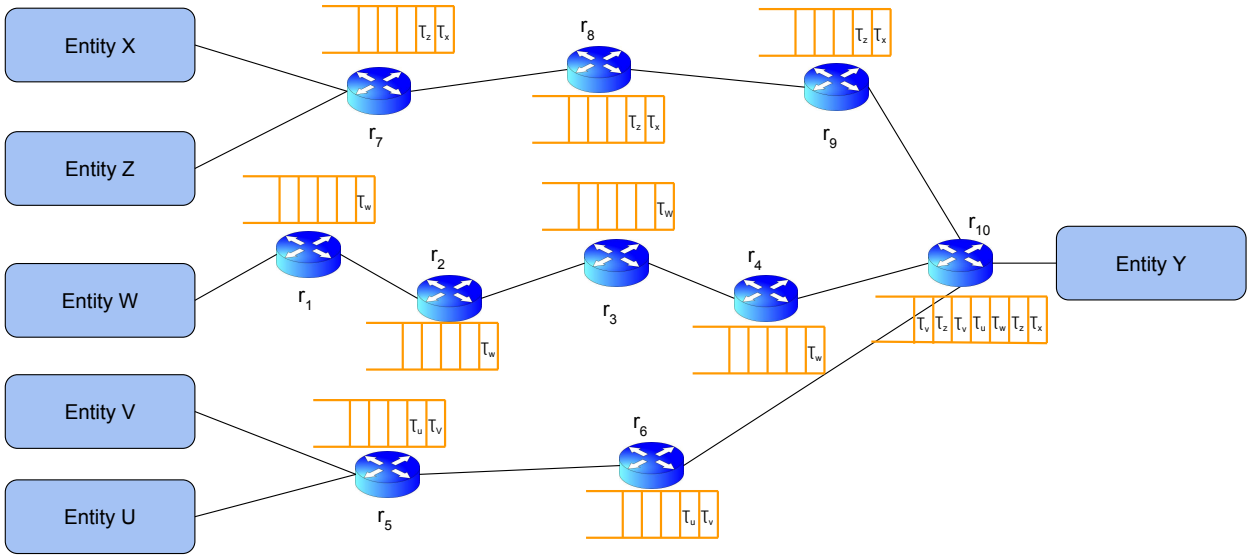

Figure A.11: An Example of Five Fog Nodes with Physical Network Connection

A.11, therefore the Gateway to calculate the queuing time for τ_Z uses the R_Z which is equal to R_X. The queuing time for τ_Z is equal to $\frac{L}{R_X}$ (or $\frac{L}{R_Z}$). Where L represents the length a tuple (all tuples have the same length L). When the Gateway in A.10 submits τ_X to the buffer of entity Y it adds $\frac{L}{R_X}$ amount of waiting time to all the tuples in the queue that came from the same path az τ_X but it only adds $\frac{L}{R_{10}}$ to the rest of tuples in the queue.

Appendix B

Query Graph

B.1 Query Graph for Congested Highway Notification Scenario

Figure B.1 illustrates a query graph that is used for the Congested Highway Notification Scenario [12].

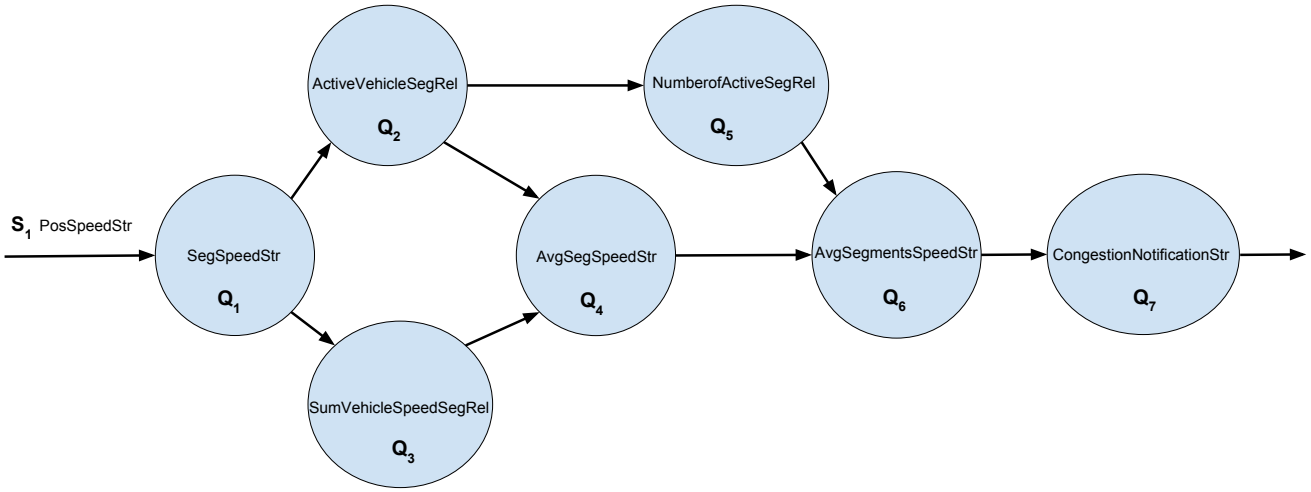

Figure B.1: Original Query Graph for Traffic Congestion Notification Scenario

- **PosSpeedStr**(*timestamp*, $longitude_info$, $latitude_info$, *dir*, *vehicleID*, *speed*, *hwy*#)**:** this is the a position report stream that is received from vehicles.
- **SegSpeedStr**(*timestamp*, *vehicleID*, *speed*, *segNo*, *dir*, *hwy*#) : This is a non-aggregation and it is like a filter that returns the segment number from longitude and latitude information from PosSpeedStr.
- **ActivieVehicleSegRel:** This returns the number of vehicles in a segment over a window of time (e.g., 3 minutes).

- **SumVehicleSpeedSegRel:** This returns the summation of vehicles' speed in a segment over a window of time (e.g., 3 minutes).
- **AvgSegSpeedStr**(*timestamp*, *avgSpeed*, *segNo*, *dir*, *hwy#*) : This returns the average of vehicles' speed in one segment over a window of time (e.g., 3 minutes).
- **NumberofActivieSegRel**(*timestamp*, *segIDs*, *dir*)**:** this returns number of active segments and number of highways that an aggregation needs to be done for them. segIDs represents the segNo and the corresponding *hwy#*.
- **AvgSegmentsSpeedStr**(*timestamp*, *avgSpeed*, *segIDs*, *dir*)**:** Calculates the average speed from all segments. So here we can aggregate from different fog nodes
- **CongestionNotificationStr**(*timestamp*, *segIDs*, *dir*)**:** If the average speed for all the segments in AvgSegmentsSpeedStr is less that a certain speed then a congestion notification will be sent.

Proposed cost functions map the given query graph to Tree 1 as following:

- C_1:
  - Original Query Graph: Q_1 is mapped to level 0 fog nodes; Q_2 and Q_3 are mapped to level 1 fog nodes; Q_4 and Q_5 are mapped to level 2 fog nodes; Q_6 is mapped to level 3 fog nodes; Q_7 is mapped to level 4 fog nodes.
  - Reconfigured Query Graph: Q_1 and Q_2 are mapped to level 0 fog nodes; Q_3, Q_4, and Q_5 are mapped to level 1 fog nodes; Q_6 is mapped to level 2 fog nodes; Q_7 is mapped to level 3 fog nodes.
  - Adjusted Query Graph: Q_1 and Q_2 are mapped to level 0 fog nodes; Q_3, Q_4, and Q_5 are mapped to level 1 fog nodes; Q_6 is mapped to level 2 fog nodes; Q_7 is mapped to level 3 fog nodes.
- C_2:
  - Original Query Graph: Q_1 is mapped to level 0 fog nodes; Q_2 is mapped to level 1 fog nodes; Q_3 is mapped to level 2 fog nodes; Q_4 is mapped to level 3 fog nodes; Q_5 is mapped to level 4 fog nodes; Q_6 is mapped to level 5 fog nodes; Q_7 is mapped to level 6 fog nodes;
  - Reconfigured Query Graph: Q_1 and Q_2 are mapped to level 0 fog nodes; Q_3 is mapped to level 1 fog nodes; Q_4 is mapped to level 2 fog nodes; Q_5 is mapped to level 3 fog nodes; Q_6 is mapped to level 4 fog nodes; Q_7 is mapped to level 5 fog nodes.
  - Adjusted Query Graph: Q_1 and Q_2 are mapped to level 0 fog nodes; $Q_4$and Q_5 are mapped to level 1 fog nodes; Q_3 is mapped to level 2 fog nodes; Q_6 is mapped to level 3 fog nodes; Q_7 is mapped to level 4 fog nodes.
- C_3:

  - Original Query Graph: Q_1 is mapped to level 0 fog nodes; Q_2 is mapped to level 2 fog nodes; Q_3 is mapped to level 3 fog nodes; Q_4 is mapped to level 4 fog nodes; Q_5 and Q_6 are mapped to level 6 fog nodes; Q_7 is mapped to level 7 fog nodes;
  - Reconfigured Query Graph: Q_1 and Q_2 are mapped to level 0 fog nodes; Q_3 is mapped to level 1 fog nodes; Q_4 is mapped to level 2 fog nodes; Q_5 is mapped to level 3 fog nodes; Q_6 is mapped to level 5 fog nodes; Q_7 is mapped to level 6 fog nodes.
  - Adjusted Query Graph: Q_1 and Q_2 are mapped to level 0 fog nodes; Q_4 and Q_5 are mapped to level 1 fog nodes; Q_3 is mapped to level 2 fog nodes; Q_6 is mapped to level 5 fog nodes; Q_7 is mapped to level 6 fog nodes.

Proposed cost functions map the given query graph to Tree 2 as following:

- C_1:
  - Original Query Graph: Q_1 is mapped to level 0 fog nodes; Q_2 and Q_3 are mapped to level 1 fog nodes; Q_4 and Q_5 are mapped to level 2 fog nodes; Q_6 is mapped to level 3 fog nodes; Q_7 is mapped to level 4 fog nodes.
  - Reconfigured Query Graph: Q_1 and Q_2 are mapped to level 0 fog nodes; Q_3, Q_4, and Q_5 are mapped to level 1 fog nodes; Q_6 is mapped to level 2 fog nodes; Q_7 is mapped to level 3 fog nodes.
  - Adjusted Query Graph: Q_1 and Q_2 are mapped to level 0 fog nodes; Q_3, Q_4, and Q_5 are mapped to level 1 fog nodes; Q_6 is mapped to level 2 fog nodes; Q_7 is mapped to level 3 fog nodes.
- C_2:
  - Original Query Graph: Q_1 is mapped to level 0 fog nodes; Q_2 is mapped to level 1 fog nodes; Q_3 is mapped to level 2 fog nodes; Q_4 is mapped to level 3 fog nodes; Q_5 is mapped to level 4 fog nodes; Q_6 and Q_7 are mapped to level 5 fog nodes
  - Reconfigured Query Graph: Q_1 and Q_2 are mapped to level 0 fog nodes; Q_3 is mapped to level 1 fog nodes; Q_4 is mapped to level 2 fog nodes; Q_5 is mapped to level 3 fog nodes; Q_6 is mapped to level 4 fog nodes; Q_7 is mapped to level 5 fog nodes.
  - Adjusted Query Graph: Q_1 and Q_2 are mapped to level 0 fog nodes; $Q_4$and Q_5 are mapped to level 1 fog nodes; Q_3 is mapped to level 2 fog nodes; Q_6 is mapped to level 3 fog nodes; Q_7 is mapped to level 4 fog nodes.
- C_3 :
  - Original Query Graph: Q_1 is mapped to level 0 fog nodes; Q_2 is mapped to level 2 fog nodes; Q_3 is mapped to level 3 fog nodes; Q_4 is mapped to level 4 fog nodes; Q_5, Q_6 and Q_7 are mapped to level 5 fog nodes.

  - Reconfigured Query Graph: Q_1 and Q_2 are mapped to level 0 fog nodes; Q_3 is mapped to level 1 fog nodes; Q_4 is mapped to level 2 fog nodes; Q_5 is mapped to level 3 fog nodes; Q_6 and Q_7 are mapped to level 5 fog nodes.
  - Adjusted Query Graph: Q_1 and Q_2 are mapped to level 0 fog nodes; Q_4 and Q_5 are mapped to level 1 fog nodes; Q_3 is mapped to level 2 fog nodes; Q_6 and Q_7 are mapped to level 5 fog nodes.

Proposed cost functions map the given query graph to Tree 3 as following:

- C_1:
  - Original Query Graph: Q_1 is mapped to level 0 fog nodes; Q_2 and Q_3 are mapped to level 1 fog nodes; Q_4 and Q_5 are mapped to level 2 fog nodes; Q_6 is mapped to level 3 fog nodes; Q_7 is mapped to level 4 fog nodes.
  - Reconfigured Query Graph: Q_1 and Q_2 are mapped to level 0 fog nodes; Q_3, Q_4, and Q_5 are mapped to level 1 fog nodes; Q_6 is mapped to level 2 fog nodes; Q_7 is mapped to level 3 fog nodes.
  - Adjusted Query Graph: Q_1 and Q_2 are mapped to level 0 fog nodes; Q_3, Q_4, and Q_5 are mapped to level 1 fog nodes; Q_6 is mapped to level 2 fog nodes; Q_7 is mapped to level 3 fog nodes.
- C_2:
  - Original Query Graph: Q_1 is mapped to level 0 fog nodes; Q_2 is mapped to level 1 fog nodes; Q_3 is mapped to level 2 fog nodes; Q_4 is mapped to level 3 fog nodes; Q_5, Q_6 and Q_7 are mapped to level 4 fog nodes.
  - Reconfigured Query Graph: Q_1 and Q_2 are mapped to level 0 fog nodes; Q_3 is mapped to level 1 fog nodes; Q_4 is mapped to level 2 fog nodes; Q_5 is mapped to level 3 fog nodes; Q_6 and Q_7 are mapped to level 4 fog nodes.
  - Adjusted Query Graph: Q_1 and Q_2 are mapped to level 0 fog nodes; $Q_4$and Q_5 are mapped to level 1 fog nodes; Q_3 is mapped to level 2 fog nodes; Q_6 is mapped to level 3 fog nodes; Q_7 is mapped to level 4 fog nodes.
- C_3:
  - Original Query Graph: Q_1 is mapped to level 0 fog nodes; Q_2 is mapped to level 2 fog nodes; Q_3 is mapped to level 3 fog nodes; Q_4, Q_5, Q_6 and Q_7 are mapped to level 4 fog nodes.
  - Reconfigured Query Graph: Q_1 and Q_2 are mapped to level 0 fog nodes; Q_3 is mapped to level 2 fog nodes; Q_4 and Q_5 are mapped to level 3 fog nodes; Q_6 and Q_7 are mapped to level 4 fog nodes.
  - Adjusted Query Graph: Q_1 and Q_2 are mapped to level 0 fog nodes; Q_4 and Q_5 are mapped to level 2 fog nodes; Q_3 is mapped to level 3 fog nodes; Q_6 and Q_7 are mapped to level 4 fog nodes;

B.2 Camera Crowd Size Measurement Scenario

Figure 6.2 illustrates a query graph that is used for measurement of crowd size in the Camera Crowd Size Measurement Scenario.

- Query Q_1 receives stream of video from camera(s) for an intersection. Query Q_1 applies an operation that returns I frames corresponding to an intersection.
- Query Q_2 returns the list of camera(s) for each intersection over a window of time.
- Query Q_3 measures the number of people for an intersection over a window of time by analysing the stream of I frames for an intersection over a window of time. Query Q_4 receives the stream of I frames from query Q_1.
- Query Q_4 calculates the average crowd size.
- Query Q_5 returns the list of intersections in a region.
- Query Q_6 calculates the average number of people for the intersections in a region over a window of time.

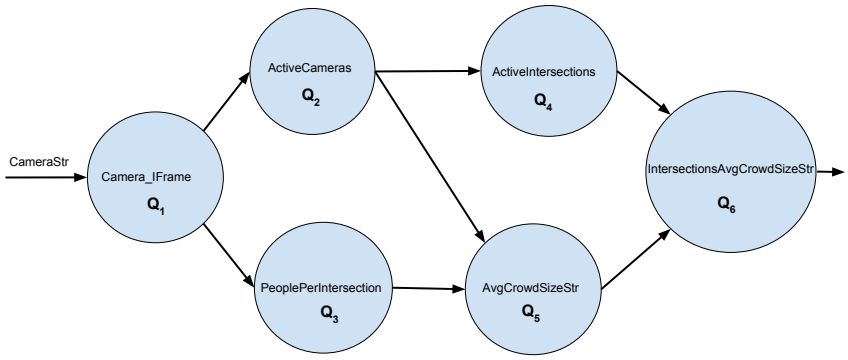

Figure B.2: Camera Crowd Size Scenario Query Graph

The proposed cost functions map the given query graph to Tree 1 as follows:

- C_1:
  - Original Query Graph: Q_1 is mapped into level 0; Q_2 , Q_3 are mapped into level 1; Q_4, Q_5 are mapped into level 3; Q_6 is mapped to level 3.
  - Reconfigured Query Graph: Q_1, Q_2 are mapped into level 0; Q_3, Q_4, Q_5 is mapped into level 1; Q_6 is mapped to level 2.
  - Adjusted Query Graph: Q_1,Q_2 are mapped into level 0; Q_3, Q_4,Q_5 is mapped into level 1, Q_6 is mapped to level 2.
- C_2:

  - Original Query Graph: Q_1 is mapped into level 0; Q_2 is mapped into level 1; Q_3 is mapped into level 2; Q_5 is mapped into level 3; Q_4 is mapped into level 4: Q_6 is mapped to level 5.
  - Reconfigured Query Graph: Q_1, Q_2 are mapped into level 0; Q_3 is mapped into level 1; Q_4 is mapped into level 2; Q_5 is mapped into level 3; Q_6 is mapped to level 4.
  - Adjusted Query Graph: Q_1,Q_2 are mapped into level 0;Q_3 is mapped into level 1; Q_4,Q_5 are mapped into level 2;Q_6 is mapped to level 3.

- C_3 :
  - Original Query Graph: Q_1 is mapped into level 0; Q_2 is mapped into level 1; Q_3 is mapped into level 2; Q_4 is mapped into level 4 ; Q_5 is mapped into level 6.
  - Reconfigured Query Graph: Q_1, Q_2 are mapped into level 0;Q_3 is mapped into level 1; Q_4 is mapped into level 4; Q_5 is mapped into level 6.
  - Adjusted Query Graph: Q_1,Q_2 are mapped into level 0; Q_3 is mapped into level 1; Q_4,Q_5 are mapped into level 5.

Proposed cost functions map the given query graph to Tree 2 as following:

- C_1:
  - Original Query Graph: Q_1 is mapped into level 0; Q_2 , Q_3 are mapped into level 1; Q_4, Q_5 are mapped into level 3; Q_6 is mapped to level 3.
  - Reconfigured Query Graph: Q_1, Q_2 are mapped into level 0; Q_3, Q_4, Q_5 is mapped into level 1; Q_6 is mapped to level 2.
  - Adjusted Query Graph: Q_1,Q_2 are mapped into level 0; Q_3, Q_4,Q_5 is mapped into level 1, Q_6 is mapped to level 2.

- C_2:
  - Original Query Graph: Q_1 is mapped into level 0; Q_2 is mapped into level 1; Q_3 is mapped into level 2; Q_5 is mapped into level 3; Q_4 is mapped into level 4: Q_6 is mapped to level 5.
  - Reconfigured Query Graph: Q_1, Q_2 are mapped into level 0; Q_3 is mapped into level 1; Q_4 is mapped into level 2; Q_5 is mapped into level 3; Q_6 is mapped to level 4.
  - Adjusted Query Graph: Q_1,Q_2 are mapped into level 0; Q_3 is mapped into level 1; Q_4,Q_5 are mapped into level 2;Q_6 is mapped to level 3.

- C_3 :
  - Original Query Graph: Q_1 is mapped into level 0; Q2 is mapped into level 1; Q_3 is mapped into level 2; Q_4 is mapped into level 4 ; Q_5 is mapped into level 5.

  - Reconfigured Query Graph: Q_1, Q_2 are mapped into level 0; Q_3 is mapped into level 1; Q_4 is mapped into level 4; Q_5 is mapped into level 5.
  - Adjusted Query Graph: Q_1,Q_2 are mapped into level 0; Q_3 is mapped into level 1; Q_4,Q_5 are mapped into level 5.

Proposed cost functions map the given query graph to Tree 3 as following:

- C_1:
  - Original Query Graph: Q_1 is mapped into level 0; Q_2 , Q_3 are mapped into level 1; Q_4, Q_5 are mapped into level 3; Q_6 is mapped to level 3.
  - Reconfigured Query Graph: Q_1, Q_2 are mapped into level 0; Q_3, Q_4, Q_5 is mapped into level 1; Q_6 is mapped to level 2.
  - Adjusted Query Graph: Q_1,Q_2 are mapped into level 0; Q_3, Q_4,Q_5 is mapped into level 1, Q_6 is mapped to level 2.
- C_2:
  - Original Query Graph: Q_1 is mapped into level 0; Q_2 is mapped into level 1; Q_3 is mapped into level 2; Q_5 is mapped into level 3; Q_4 and are mapped into level 4.
  - Reconfigured Query Graph: Q_1, Q_2 are mapped into level 0; Q_3 is mapped into level 1; Q_4 is mapped into level 2; Q_5 is mapped into level 3; Q_6 is mapped to level 4.
  - Adjusted Query Graph: Q_1,Q_2 are mapped into level 0; Q_3 is mapped into level 1; Q_4,Q_5 are mapped into level 2;Q_6 is mapped to level 3.
- C_3 :
  - Original Query Graph: Q_1 is mapped into level 0; Q_2 is mapped into level 1; Q_3 is mapped into level 2; Q_4 and Q_5 is mapped into level 4.
  - Reconfigured Query Graph: Q_1, Q_2 are mapped into level 0; Q_3 is mapped into level 1; Q_4 and Q_5 are mapped into level 4.
  - Adjusted Query Graph: Q_1,Q_2 are mapped into level 0; Q_3 is mapped into level 1; Q_4,Q_5 are mapped into level 4.

Curriculum Vitae

Name:	Navid Bayat
Education and Degrees:	Department of Computer Science 2012 - 2018 Doctor of Philosophy University of Western Ontario London, ON
Related Work Experience:	Teaching Assistant The University of Western Ontario 2012 - 2016 Research Assistance The University of Western Ontario 2012 - 2018

Publications:

1. Bayat, N.; Lutfiyya, H. Coping with Flash Crowd in Multi-Channel Live Video Streaming for Peer-to-Peer Networks. In: *Proceedings of Third International conference on selected topics in Mobile and Wireless Networking (MoWNet)*, August 2013.

2. Bayat, N.; Lutfiyya, H. MC-SkyNet: Mobile-Cloud Dynamic Partitioning for Mobile-Cloud Applications.In: Proceedings of The International Workshop on Distributed Mobile Systems & Services (DMSS), June 2014.

3. Bayat, N.; Lutfiyya, H. Network Coding for Coping with Flash Crowd in P2P Multi-Channel Live Video Streaming.In: Proceedings of The International Conference on the Design of Reliable Communication Networks (DRCN), March 2015.